

2010

An Adaptable Group Communication System

Vikram Reddy Kayathi

Louisiana State University and Agricultural and Mechanical College, vkayat1@tigers.lsu.edu

Follow this and additional works at: https://digitalcommons.lsu.edu/gradschool_theses



Part of the [Computer Sciences Commons](#)

Recommended Citation

Kayathi, Vikram Reddy, "An Adaptable Group Communication System" (2010). *LSU Master's Theses*. 180.
https://digitalcommons.lsu.edu/gradschool_theses/180

This Thesis is brought to you for free and open access by the Graduate School at LSU Digital Commons. It has been accepted for inclusion in LSU Master's Theses by an authorized graduate school editor of LSU Digital Commons. For more information, please contact gradetd@lsu.edu.

AN ADAPTABLE GROUP COMMUNICATION SYSTEM

A Thesis

Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in System Science

in

The Interdepartmental Program in
The Department of Computer Science

by

Vikram Reddy Kayathi

B. E., Chaitanya Bharathi Institute of Technology, Osmania University, 2008
December 2010

Acknowledgements

I am very grateful to my advisor Dr. Supratik Mukhopadhyay for his guidance, patience and understanding throughout this work. His suggestions, discussions and constant encouragement have helped me to get a deep insight in my thesis work. I would like to thank Dr. S. S. Iyengar and Dr. Konstantin Busch for sparing their time to be a part of my thesis advisory committee. Also I would like to thank Dr. Sukhamay Kundu for providing valuable inputs in my thesis work. Also I am very thankful to Department of Computer Science Department.

Also I'm very thankful to the scalaris developer's team who helped me a lot in my work. I'm also very thankful to my friend Gaurav Mangukiya who helped me.

I wish to endow my earnest gratitude to my parents, who believed in me and have been thorough all the rough times. I also want to thank my entire family and friends for their affection, support and compassion.

Table of Contents

Acknowledgements.....	ii
Abstract.....	iv
1. Introduction	1
2. Requirements: Mission Critical Systems Vs E Commerce Systems	3
3. Existing Group Communication Systems	9
4. Design and Development of Reliable Group Communication System over Asynchronous State	16
5. Algorithms used in implementation of GComm	29
6. Related Work	50
7. Experimental Results	53
8. Conclusions	61
References.....	62
Vita	64

Abstract

Existing group communication systems like ISIS, Spread, Jgroups etc., provide group communication in a synchronous environment. They are built on top of TCP/IP or UDP and guarantee virtual synchrony and consistency. However, wide area distributed systems are inherently asynchronous. Existing group communication systems are not suitable for wide area deployment. They do not provide persistent communication; i.e., if a node gets temporarily disconnected, all messages directed to that node during that period are lost. Hence such systems are not suitable for deployment in disadvantaged networks.

While, according to Brewer's CAP theorem, it is impossible for a distributed computer system to achieve the three objectives of consistency, availability and partition-tolerance simultaneously, especially in an asynchronous environment, we present the design and development a reliable group communication system over an asynchronous substrate, where we achieve the objectives of eventual consistency, availability and partition-tolerance. We say that distributed system is eventually consistent, over a long period of time, if no updates are sent and all updates will eventually propagate through the system and all the nodes will be consistent. By availability we mean that if a node failure does not prevent the system to operate continuously. We say that distributed system is partition-tolerant if in the event of network failure that splits the processing nodes which are communicating, then the system should allow the processing to continue in both subgroups.

1. Introduction

Ever since the invention of computers, the communication between them became a necessity. The purpose of communication may vary from message passing to file sharing. They communicate through various communication channels in a network. A distributed system consists of many computers which communicate through a channel in a network. A group communication system is necessary for a distributed system, as it provides a communication layer between the sender and receiver, and ensures that messages are delivered accordingly with synchronous or asynchronous semantics.

According to Brewer's CAP theorem, it is impossible for a distributed system to achieve all the three objectives: consistency, availability and partition tolerance simultaneously in an asynchronous environment [4]. Distributed systems are inherently asynchronous, where the sender of messenger will not wait for the receiver to be ready. The existing group communication tools provide the communication in a synchronous semantics achieving the virtual synchrony and consistency, but not efficient in asynchronous environments.

To provide a group communication for asynchronous distributed systems, we designed and developed a reliable group communication system over an asynchronous substrate. The group communication tool being GComm and the asynchronous substrate is the distributed database, scalaris [11]. As it is impossible to achieve all the three objectives of Brewer's CAP theorem, we relaxed one of the objectives, consistency with eventual consistency. Our group communication system achieves the three objectives: eventual consistency, availability and partition tolerance, and provides no-message loss and uses

continuation passing to maintain the persistence. We implemented publish-subscribe framework in our group communication system.

This rest of thesis is organized as follows: Chapter 2 discusses the example of a scenario which motivated us for design and development of our work, and compares the requirements of mission critical systems and e-commerce systems with respect to group communication. Chapter 3 explains some of the existing group communication tools and their problems. Also it discusses about the Brewer's CAP theorem and the relaxation of the CAP theorem by replacing consistency with eventual consistency [3]. Chapter 4 explains our design and development of reliable group communication system over an asynchronous substrate along with the UML diagrams. Chapter 5 explains the series of algorithms used in the implementation of the GComm. Chapter 6 covers the related work we did prior to design and development of our system. Chapter 7 provides the experimental results achieved when we implemented our work. Finally, chapter 8 concludes our work and discusses the future work.

2. Requirements: Mission Critical Systems Vs E Commerce Systems

2.1 Motivating Example

The motivating example for our work is the series of the events which happens after an accident crash in USA. After the accident, the node *person* who is involved in accident will notify the emergency system by publishing the crash information like location, causalities, need of ambulance services, need of fire services etc., The node, *person* publish this information in the transactional storage database by joining the group 1, then the node *911* subscribes to the information published by *person* by joining the group1.

There are various other groups in the emergency system involving the nodes *911*, *cops*, *ambulance services*, *fire services* etc., The nodes *911* and *cops* are members of group 2, the nodes *911* and *ambulance services* are members of group 3, the nodes *911* and *fire services* are members of group 3.

After *911* subscribed to the crash information, it publishes the information to the transactional storage database. Then the node *cops* will subscribe to the information published by *911*, then react accordingly. Similarly, if there is a need of ambulance services and fire services then they subscribe to information published by *911* and react accordingly. If the crash involves any health emergencies and fire accidents which require attention of fire emergency services, then the node *911* will publish the required information to the transactional storage database, through which the nodes *fire services* and *ambulance services* will subscribe to the database and respond accordingly.

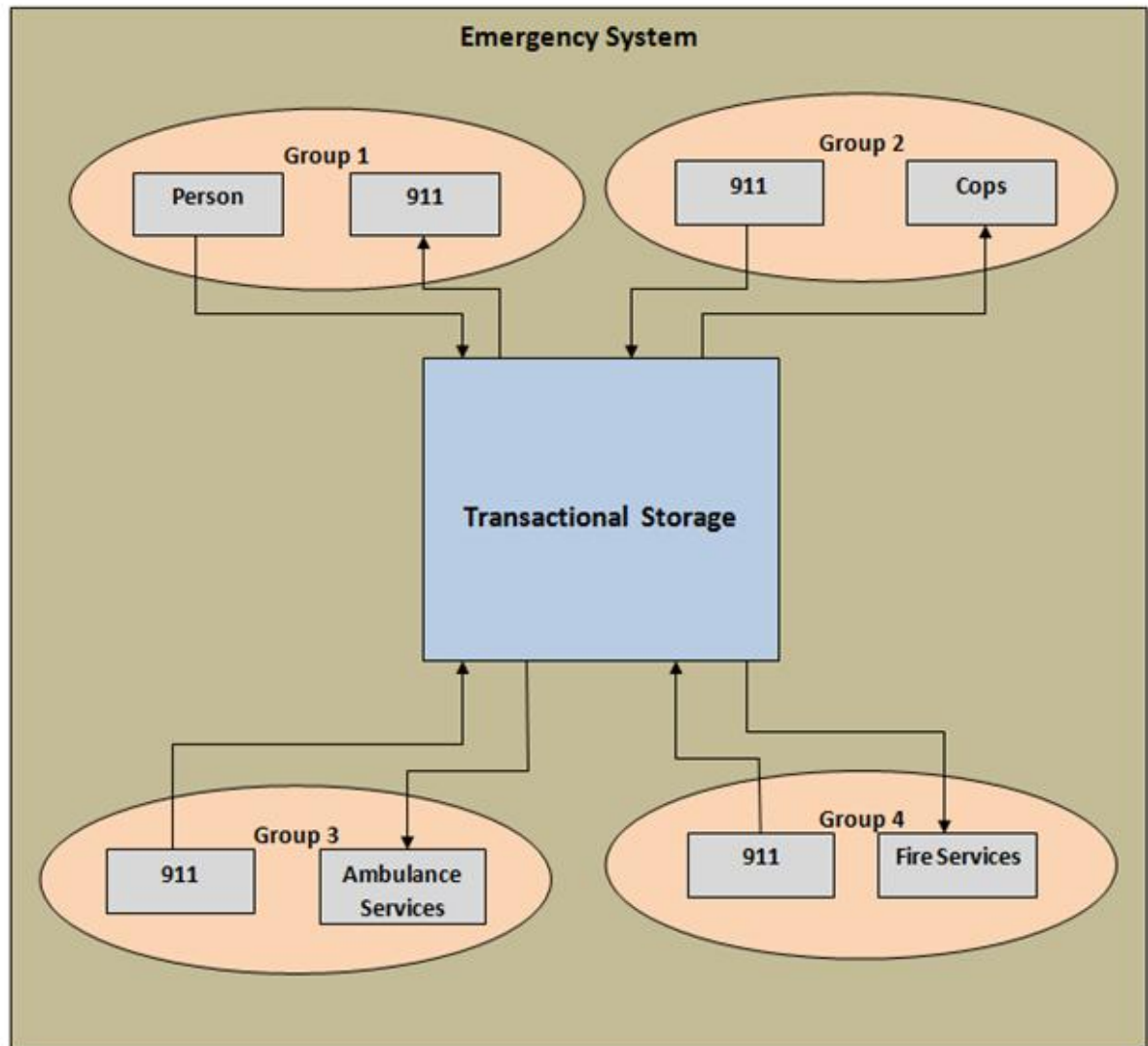


Figure 2.1 Emergency system consisting various groups

2.2 Requirements: Mission Critical System Needs Vs E-Commerce System Needs

Mission critical system means if any of activity, module, functionality or system fails then the whole business operation fails. If any business organization has to remove any constraints or modules for some reason, the constraint or module should not be mission critical, as the success of the project purely depends on the functionality of them. Electronic commerce systems, also referred as e-commerce systems comprises of the purchasing and selling the products of services over the internet or any other computer networks. Companies like

Amazon, EBay etc., implement the e-commerce systems. The selling and buying are implemented electronically in a real time environment. Both systems i.e., mission critical system and e-commerce systems need to be operated in distributed computing environment. Hence the nodes or computers in both the systems need to communicate with each other through group communication system. So, we have compared the requirements of the mission critical systems and e-commerce systems with respect to group communication. We compared both of the systems with respect to different group communication attributes like robust connectivity, sequential consistency, high availability, partition tolerance , churn tolerance, mutable objects, real time response, relational operations, data persistence and large data.

2.2.1 Robust Connectivity Over Disadvantaged Links

Robust connectivity over disadvantaged networks means, the connectivity between the nodes or computers in distributed computing environment always exists even though there is a failure in the network for any reason. Usually, networks fail frequently either with power disconnection or node failure, such networks are called disadvantaged networks. Mission critical systems require robust connectivity over disadvantaged links, as the connection failure might result in the whole system failure. While e-commerce systems doesn't require the robust connectivity because there may be connection failure at the end user in the process, and still the transaction can be carried out once the end user reconnects.

2.2.2 Sequential Consistency

Sequential Consistency means "the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program." [\[1\]](#)

The system is said to be sequentially consistent if all the operations carried out are in an order and every read/write operation carried out by the node is clearly visible throughout the system for every node in the system. Both mission critical systems and e-commerce systems need the attribute sequential consistency.

2.2.3 High Availability

The term availability means, the group communication system should be almost always be up and continue to communicate between the nodes, even though there is a node failure. It shouldn't prevent the other nodes to communicate with each other. Both mission critical systems and e-commerce systems require the high availability attribute.

2.2.4 Partition Tolerance

A group communication system is partition tolerant "if there is a network failure that splits the processing nodes into two groups that cannot talk to each other, then the goal would be to allow processing to continue in both subgroups" [3]. The system continues to operate despite arbitrary message loss. Mission Critical systems strongly require the partition tolerance, while e-commerce systems are less stringent towards having the partition tolerance requirement.

2.2.5 Churn Tolerance

A group communication system is churn-tolerant if the nodes can join into the system and leave the system at any point of time. The nodes inlet and outlet in the group communication system shouldn't affect the functionality of the other participating nodes. Mission Critical systems strongly require the churn-tolerance, while e-commerce systems are less stringent towards having the churn tolerance requirement.

2.2.6 Mutable Objects

The objects in the group communication system are mutable, if the objects can get modified and again join the system at any point of time. Both mission critical systems and e-commerce systems require the mutable objects in their group communication system.

2.2.7 Real Time Response

The operation in the group communication system should be implemented in real time environment. Mission critical systems require real time response, where there will be a definite time to react on a given input. While e-commerce systems may not need real time response from the group communication system.

2.2.8 Relational Operations

Relational operations are database tables which have data ordered and organized on the basis of different common characteristics in the form of tables. Mission critical systems usually don't require the relational operations. While e-commerce systems usually require relational operations between the data as there will be a large amount of data associated with the group communication system.

2.2.9 Data Persistence

Data persistence means, the data will be made available even though the process terminates. As the Mission critical systems don't deal with large data, they don't require the data persistence. On the other hand, e-commerce systems require the data persistence for their group communication system.

2.2.10 Large Data

Mission critical systems don't require large data; on the other hand e-commerce systems require large data for their group communication.

3. Existing Group Communication Systems

Group communication systems provide the communication medium between the nodes/users in a network. They are implemented over synchronous and asynchronous networks. In a synchronous network, the sender sends the message to the receiver only when the receiver is ready to receive it i.e., the sender waits for the receiver to be ready. In an asynchronous network, the sender sends the message to the receiver and will not wait for the receiver to be ready. A group communication system provides a communication layer between the sender and receiver, and ensures the messages are delivered accordingly with synchronous or asynchronous semantics. A group communication tool has a very effective implementation of constructing replica system. They are generally implemented over distributed database. There are some prominent group communication tools such as:

- ISIS (Birman et. Al.)
- Spread (Amir et. Al.)
- Jgroups (Commercial implementation in Java underlying the Jboss middleware)

The above tools provide group communication in a synchronous environment. They are built on the top of TCP/IP or UDP protocols. The key features they provide are virtual synchrony and consistency. Virtual synchrony is a property that allows the nodes to form the process groups to receive the messages. Every node in the process group receives the message sent to the process group which they belong and in the same order even though they receive more than one message. The uses of the virtual synchrony are data replication, fault tolerance, event notification and caching. All the nodes in a network are consistent when there are any write operations; it is clearly visible in the network to the other nodes. All nodes will see the

messages in the same order. Either all nodes receive a message or none. If nodes are in a group, then if one node receives a message then each node in that process group will receive and if any one node in a process group will not receive a message then, no other node in the process group will receive the message. The goal is to have all or nothing semantics.

3.1 ISIS

The ISIS group communication tool is developed in Cornell University. It implements the virtual synchrony for its group communication system. There are four different process groups implemented in ISIS, they differ in their implementation of interaction with the groups. The four groups are: peer groups, client groups, diffusion groups and hierarchical groups [5] [6].

- Peer groups comprise of processes which contain replicated data given as input to algorithms processing concurrent data.
- Client groups contain the nodes which try to communicate with any process group with a group name and proper authorization, and then the process group makes that node as a client to the group by registering it with the group.
- Diffusion groups are groups of nodes depicting the client server architecture. Client nodes interact with server nodes by giving input and getting desired output from the server nodes.
- Hierarchical groups contain one or more process groups. There will be one base group root which will have other groups called sub groups under it.

ISIS nodes may or may not aware of one another. ISIS implements message delivery ordering rather than implementing the casual relationship between messages. As mentioned

earlier, multiple modules are allowed to form a group under a group name and any message transmitted to the group will be received by all the nodes in that group.

3.2 Spread Group Communication Toolkit

The Spread wide area group communication system (Amir et. Al.) is developed in John Hopkins university. Spread comprises of two low level protocols, ring and hop. Ring is implemented on local area networks and hop is implemented on wide area networks. And it implements daemon-client architecture. In this architecture, the group membership updates are done with minimal effort. When any node joins and leaves the group, the whole process is translated into a single message. But when there is network partition between the nodes in local area network, the update causes a fully fledged change in node's group membership. Extended virtual synchrony is implemented in spread group communication system; it means that the messages are transmitted despite the loss of messages using a variant of the alternating best protocol. Data is transmitted to the network comprising of necessary minimal set of components. User have control over the spread group communication system, they can send a message with priority over other messages transmitted in the network. Another prominent feature of this group communication system is that any node which is not a member of the group can transmit the message to the whole group [8].

3.3 Jgroups

Jgroups is the commercial implementation of a group communication system in java underlying the Jboss middleware. Jgroups is a group communication system which implements the reliable multicast communication. In Jgroups, groups containing nodes can be created and

deleted. The nodes are spread across local area networks and wide area networks. And when every node joins or leaves the group, the update is notified to every node in the group. The messages between nodes are of two types: node to node and node to group. Jgroups can implement different protocols like User Datagram Protocol (UDP), Transmission Control Protocol (TCP) and Java Message Service (JMS). Large messages are subjected to fragmentation and are encrypted when required. If there is any message loss, then the message is retransmitted. One more important feature of Jgroups is failure detection; the crashed nodes are removed from the group [7].

3.4 Problems with Existing Group Communication Systems

- Distributed systems are inherently asynchronous. In distributed systems, the sender sends the message to the receiver and will not wait to send the next message until the receiver receives the first message. Distributed systems have nodes which are across over local area networks (LAN) and wide area networks (WAN).
- The existing group communication systems which are discussed earlier are synchronous, where the sender sends the message to the receiver and will wait to send the next message until the receiver receives the first message and send the acknowledgement.
- Existing group communications systems which are synchronous are not suitable for asynchronous environments.
- The existing group communication systems are centralized. They have a single point of failure i.e., failure of one node can cause failure of the whole system.

- The existing group communication systems will have components which are not completely aware of other components in the system. For example, In ISIS the nodes in the system may or may not know the other nodes in the system.
- Whenever a component gets disconnected, all the messages directed to it are lost during the disconnection. For example, Spread group communication system experiences the loss of messages due to component failure and various reasons.
- Only two of Brewer's CAP theorem attributes are implemented in the existing group communication systems which are discussed earlier.

3.5 Brewer's CAP Theorem

According to the Brewer's CAP theorem, it is not possible for a distributed system to achieve all the three attributes mentioned below simultaneously especially in an asynchronous environment [4]:

- Consistency
- Availability
- Partition Tolerance

3.5.1 Consistency

A group communication system containing the nodes in a distributed computing environment is said to be consistent when it promises to have the familiar all-or-nothing semantics, commonly supported by group communication systems is achieved. Stronger requirements can include no message reordering. In addition, each node in the system should be consistent. The nodes in the system are said to be consistent if

- All nodes in the system will see the message in the same order.
- Either all nodes in the group will receive the message or none of them will receive.
- When a node leaves or join the group communication system, every node which is already a member of the system should be notified of the update.

3.5.2 Availability

A group communication system is said to be highly available when the system is almost always up even though the failures occur. The failures may be a node failure, module failure, component failure etc., The node failure should not prevent the survivors continuing its operations. If there is a failure in the node, then the system should switch to another node in order to keep the systems running [3].

In other words, a group communication system shouldn't be a centralized system, where the single point failure occurs. In a centralized group communication system, when there is any failure in node or component, then the whole system is down. For availability purposes, it should not be a centralized system rather it should be decentralized system.

3.5.3 Partition Tolerance

A partition happens when there is occurrence of network failure resulting in a communication gap between the two nodes in a group. A group communication system is said to be partition-tolerant whenever there is any split in the communicating nodes due to network failure then the aim of the system should be to allow the processing to continue in the both subgroups [3]. In a group communication system, there will be three types of communication: node to node, node to group, group to group.

So when there is a failure in the network which may cause interruption in the communication between the groups, the individual groups should continue to operate independently. This situation may experience the arbitrary message loss but the system continues to operate.

3.5.4 Eventual Consistency

As mentioned earlier, according to Brewer's CAP theorem it is impossible for distributed systems to provide the three objectives: consistency, availability and partition tolerance simultaneously in an asynchronous environment. We relaxed one of the objectives: consistency, as the other two objectives cannot be compromised by the distributed system in an asynchronous environment. Hence, we relaxed the objective consistency and replaced it with eventual consistency.

Eventual Consistency means, "Over a long time period where no updates are sent, we can expect that during this period, all updates will eventually, propagate through the system and all the nodes will be consistent" [3].

4. Design and Development of Reliable Group Communication System over Asynchronous State

To achieve the three objectives, Eventual Consistency, Availability and Partition Tolerance, a Reliable group communication system over a distributed database is developed. We use the distributed database scalaris as our substrate. Our group communication tool is GComm.

Distributed systems are inherently asynchronous, where the sender will not wait for the receiver to be ready. As the existing group communication systems are suitable for synchronous environment, doesn't implement the Brewer's CAP theorem, centralized and suffer from a single point of failure. Our system implements the Brewer's CAP theorem by relaxing one of its objective mentioned, consistency. We relaxed the attribute consistency with the Eventual Consistency (Michael Stonebrecker et. Al.). The system is eventually consistent if for a long period of time, there are no updates sent, we expect that all the updates will go through the system and all the nodes will be consistent during this period. When a node joins or leaves the network, the system will not be consistent at that particular moment. But when there is no update of node joining and node leaving the group for a long time period, then all the nodes will become eventually consistent in this time period [3].

Our system always continues to operate despite of the node failures or component failures, which makes the system available all the time. If there is a node failure, then the system switches to another node and continues to run as before. It never lets the survivor nodes to abort the operation when there are failures. If our system suffers a network failure which frequently happens in a disadvantaged network, then it will not abort the

communication between the groups. Instead, it allows the operations in the sub groups individually, and it continues to operate despite the arbitrary message loss which makes the system partition tolerant.

Our work, a reliable group communication system over asynchronous substrate achieves all the three objectives mentioned below:

- Eventual Consistency
- Availability
- Partition Tolerance

Apart from the three main objectives mentioned above, our system has mutable objects which mean the values of the objects can be changed as desired. Nodes can join and leave the group at any point of time without any restriction making the system churn-tolerant. When the nodes frequently join and leave the group, the system will not be eventually consistent in that time period, as the updates sent to the system continuously. The real time response objective is achieved in the system.

4.1 Group Communication System (GComm)

A Group Communication System (GComm), a group communication system which is suitable for asynchronous and disadvantaged networks. GComm achieve the three main objectives: eventual consistency, availability and partition tolerance, along with other objectives churn tolerance, mutable objects and real time response. It uses the continuation message passing to maintain the data persistence in the disadvantaged networks, which have frequent failures in the network. GComm guarantee the objectives: no message loss and no

message reordering. GComm consist of none or more nodes grouped under a group. A group with a unique name has one or more nodes as its members. The definition of the group and node are discussed in next section.

GComm uses publish-subscribe framework in its infrastructure. According to publish-subscribe framework, the publisher publishes the topic, on which values are written with a sequence number associated with them. If the subscriber wants to read those values, it has to subscribe to the topic with the time period of subscription called lease. The publisher and subscriber join the group as the members of the group. When subscriber finishes reading the values of the topic published, it issues the delete command to the system. The system verifies if all the subscribers to a particular topic issued the delete command and make sure the lease period is expired to delete the values of the topic published. A topic can be subscribed by none or more subscribers. The system tracks the subscriber's last read sequence number and its state in order to continue from the same instant when subscriber gets disconnected due to any reason like network failure.

4.1.1 Definition of a Group

A group is an environment, consisting of one or more nodes communicating with each other. Other features of the group are as follows:

- Group is a non-empty set of nodes, with a unique name which is also its identifier.
- A group is initiated by a client by registering its name with the GCS system, and this client immediately becomes the first member of the group.

- The membership relation between clients and groups is many to many, which is onto the groups but need not be onto the clients.
- Two or more groups can overlap i.e., two or more groups can consist one or more nodes in common (as their members).
- Group becomes nonexistent if there are no nodes in it. The group communication system, GComm will delete the groups as soon as they become nonexistent.
- Any number of nodes can join or leave the group at any point of time. A group may also shrink when a member (which may be the creator of the group) exits the group and when a group becomes empty, it is deleted from the GCS
- Each member of group has a view of that group.
- Each node will have view for each group it is member of.

Every group in the system has following attributes:

- Group name
- Members list -> it contains the list of the nodes which are currently in the group.
- Readers list -> Readers list is subset of the members list. Whenever there is an update of node joining or leaving the group, the nodes in the members list will read the update and add themselves to the reader list.
- Changemakers list -> it contains the list of nodes which induces the update.
- State of group (Stable/Unstable) -> the group is stable, if reader list is equal to the members list and vice versa.

Scenario: A node N1 creates a group G1, and then Node N2 joins G1. Node N1 leaves the group, and then Node N2 leaves the group. The group G1 which don't have any nodes in it becomes nonexistent and the system deletes the group.

4.1.2 Definition of a Node

A node is a member of a group, which communicates with the other nodes in the group.

Every node has a node configuration file, which has details of:

- Node name
- Group name, which is to be joined.
- IP address of the database

The node configuration file is created by node admin. Node admin can change the node configuration file, if it wants the node to join other groups in a system by adding those group names to the file.

- When Node joins or leaves the group, it updates its view.
- Every node can be a member of one or more groups. If a node N1 is member of groups G1 and G2, G1 is stable and G2 being unstable. Then N1 is said to be stable with respect to G1.

4.1.3 GComm Architecture

- In GComm infrastructure, all the nodes share the data randomly.
- The agents run on the different nodes of the network are programmable. The agents subscribe to transactional storage and generate action

- As the agents subscribe to the transactional storage, they publish on the network,
- The agents can be deployed automatically or manually.

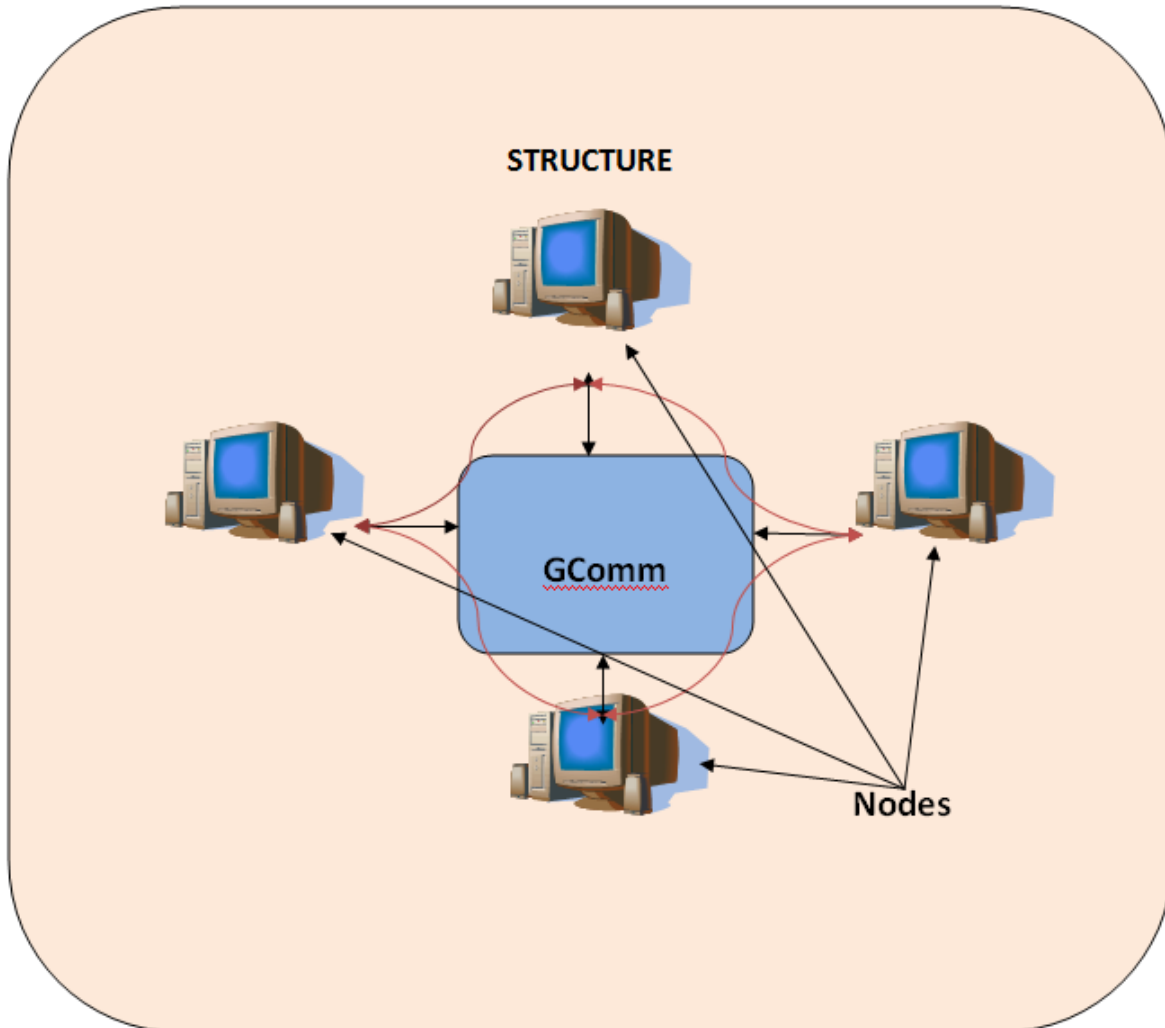


Figure 4.1.3.1 A Structure of GComm consisting of the nodes

- Every Node is aware of the presence of other nodes facilitating the data replication in the system.
- The individual process running on different machines interact through GComm.
- Nodes can join or leave the group at any point of time.
- Each node can be a member of one or more groups in GComm.

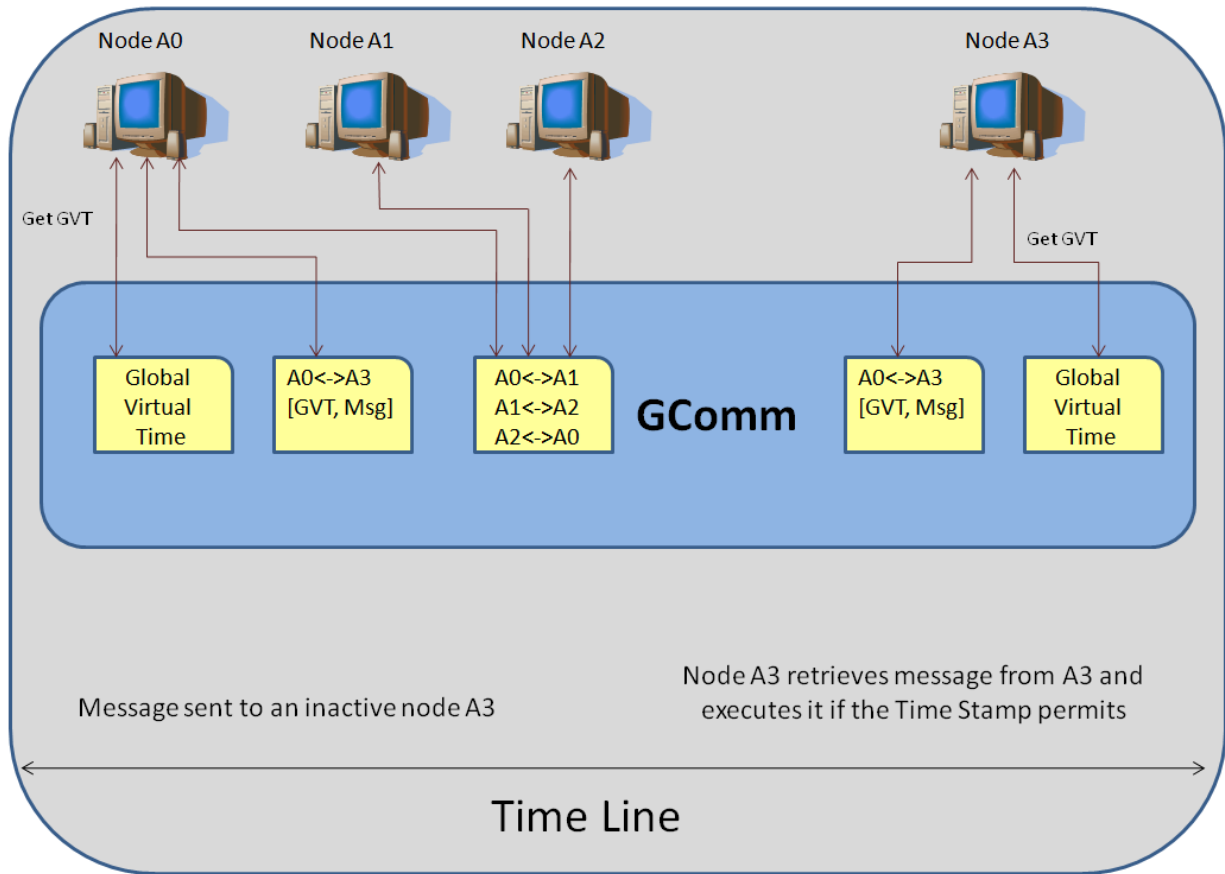


Figure 4.1.3.2 Message Passing in Gcomm consisting of four nodes, where message is passed from Node A0 to Node A3

If the nodes in the GComm are located in different time zones, there may be a conflict in time stamps of the messages. To prevent that conflict, Mattern's GVT Algorithm [17] is implemented. According to the Mattern's GVT algorithm, the nodes which have a time conflict between them will participate in a token ring procedure, in which any of participating node's time is selected as leader election time. The leader node's time will be the standard time for the procedure, which will be called as Global Virtual Time (GVT) [17].

In the above figure, there are nodes A0, A1, A2 and A3 are in GComm environment. Node A0 sends a message to node A3 which is inactive. When the message is transmitted from the A0, the message is time stamped with the GVT. As individual process running on different

machines interact through GComm, nodes A1 and A2 are also notified of the process between nodes A0 and A3. When node A3 becomes active, it checks its messages and the receiving time. It checks the current time and processes the message sent from node A0 if it is valid.

4.1.4 Application Programming Interface

```
Seq = publish(Topicname, Value);  
Id = subscribe(Topicname, Lease);  
<Value, Seq> = read(Topicname, Id);  
Delete(Topicname, Id, Seq, State);  
<State, Seq> = subscribe(Topicname, Lease, Id);
```

- Seq = publish(Topicname, Value);

When the publisher publishes the topic and value on top of it, the GComm system will provide the sequence number to the corresponding value.

- Id = subscribe(Topicname, Lease);

When subscriber subscribes to the topic by providing the topic name and lease period, the GComm system will provide the subscriber ID to it.

- <Value, Seq> = read(Topicname, Id);

When the subscriber reads any value on the topic to which it subscribed with topic name and subscriber ID, the GComm will update the value as last read value and sequence number as last sequence number.

- Delete(Topicname, Id, Seq, State);

Subscriber issues the delete command when it finishes reading the values by providing the topic name, ID, sequence number and the state.

- `<State, Seq> = subscribe(Topicname, Lease, Id);`

If a subscriber wants to reconnect and read the values of topic published, then it will provide topic name, lease and ID. The GComm verifies the subscriber lease period, if it is in the lease then it updates the state of subscriber and provides the sequence number, which is incremented by one to last read sequence number. Else, it updates the state and sequence number as last read sequence number.

4.2 Scalaris

Many global businesses comprising of e-commerce platforms deploy the distributed computing which require highly efficient concurrent access to it. There will be several millions of the read operations which conflicts with the write operations; they have to be done in a fraction of seconds. Enterprises like Amazon, Ebay or Google tackle these problems by deploying several thousands of the servers in the distributed data centers. But the problem arises in maintaining the consistent state hiding the failures from the application, as there is a failure in components frequently. Till now peer-to-peer protocols have been providing the self management among the peers which allows only write once and read many type of data sharing. The missing feature of the peer-to-peer protocol is the support of consistent replication and fast transactions which is a substantial quality [10].

To overcome the disadvantages peer-to-peer facing, scalars is developed. Scalaris, a scalable, distributed key/value store. Scalaris is built on a structured overlay network and uses a distributed transaction protocol, both of them implemented in erlang with an application interface in Java [10]. It uses Lamport's paxos algorithm [18] to maintain the consistency and uses primary memory storage. The paxos algorithm makes sure that every node is consistent. The paxos algorithm tries to bring the consensus among the nodes in a system which have a

conflict in choosing a value. There may be in delay of messages because time will be taken to bring the consensus among the node.

Also, the scalaris implements the ACID properties on the structured overlay network mentioned above. To prove the efficiency of the scalaris , it is implemented with a simple Wikipedia at the front end and the scalaris database on the backend. As the wikipedia, requires several thousands of read operations and write operations which are to be done concurrently maintaining the consistency, it is ideal to prove the efficiency of the scalaris [10].

4.2.1 Scalaris System Architecture

As mentioned earlier, scalaris is a distributed key/value store which is built on the structured peer-to-peer overlay that allows consistent read/write operations. The system architecture comprises of three layers, as follows [10]:

1. P2P Layer
2. Replication Layer
3. P2P Layer

The bottom layer is a structured P2P overlay network which performs logarithmic routing and builds the basis for the key/value store. This network stores the keys in lexicographical order unlike many distributed hash tables. The middle layer as the name suggests, it implements the replication and transaction, i.e., ACID properties (Atomicity, Concurrency, Isolation, Durability) for the concurrent write operations. And the top layer hosts the real application, a distributed key/value store. This layer can be used as a scalable, fault-tolerant backend for online services for shopping, banking, data sharing or social networking websites [10].

4.3 UML Diagrams

This section consists of UML diagrams i.e., use case diagram and state diagram mentioning the activities of the node in a group in GComm. As mentioned earlier, the group in a group communication system GComm is a non-empty set of nodes, with a unique name which is also its identifier. Two or more groups can overlap i.e., two or more groups can consist one or more nodes in common (as their members). Group becomes nonexistent if there are no nodes in it. Any number of nodes can join or leave the group at any point of time. A group may also shrink when a member (which may be the creator of the group) leaves the group and when a group becomes empty, it is deleted from the GCS. A node is a member of a group, which communicates with the other nodes in the group. Every node has a node configuration file, which has details of: node name, group name, which is to be joined and IP address of the database.

A node in a group communication system GComm has following activities:

- A node can create a group, if it doesn't exist.
- A node can join a group.
- A node can read the state of the group.
- A node can publish a topic.
- A node can subscribe to a topic
- A node can leave the group.

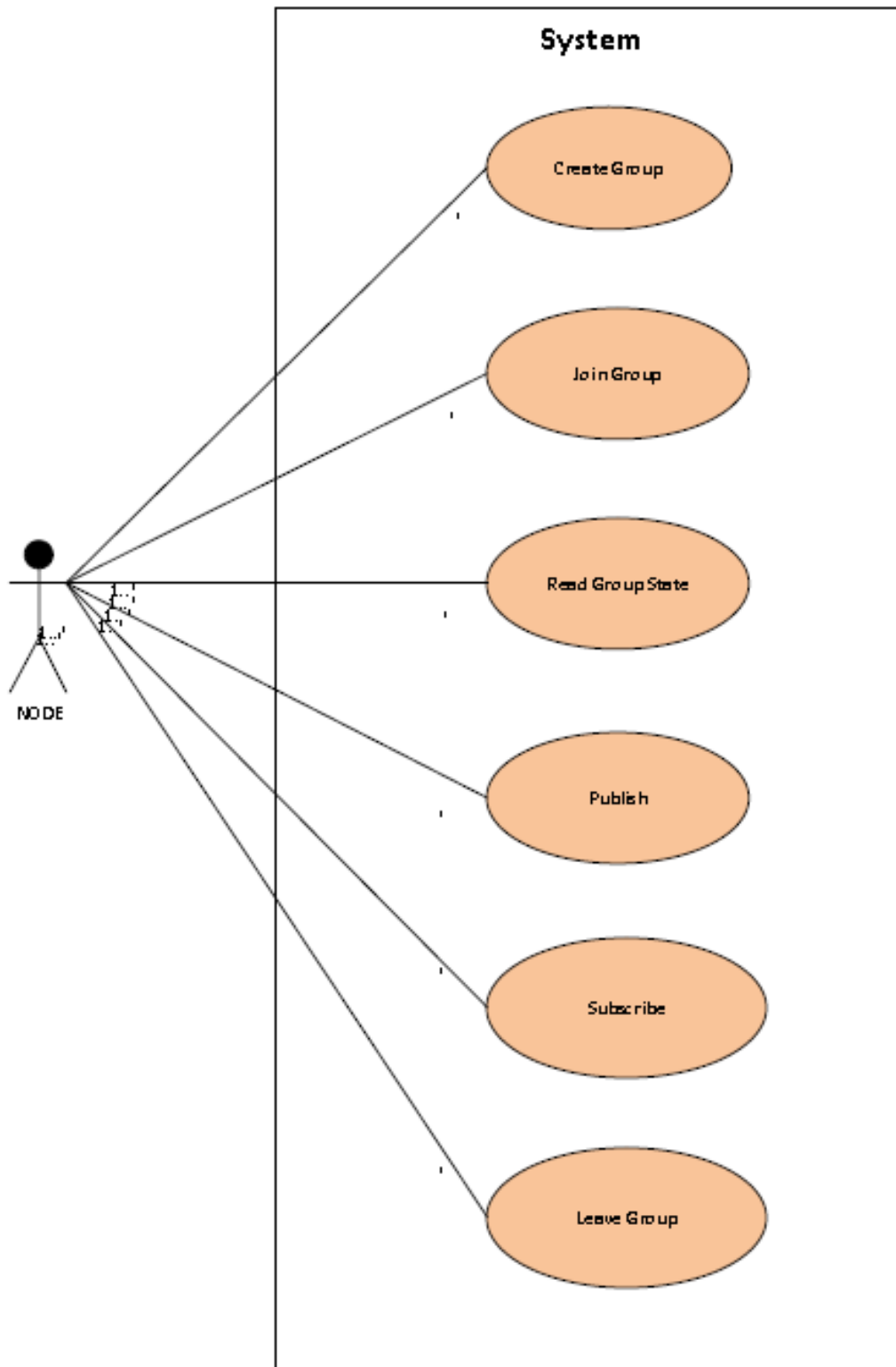


Figure 4.3 A use case diagram displaying the node and its activities in a group communication system.

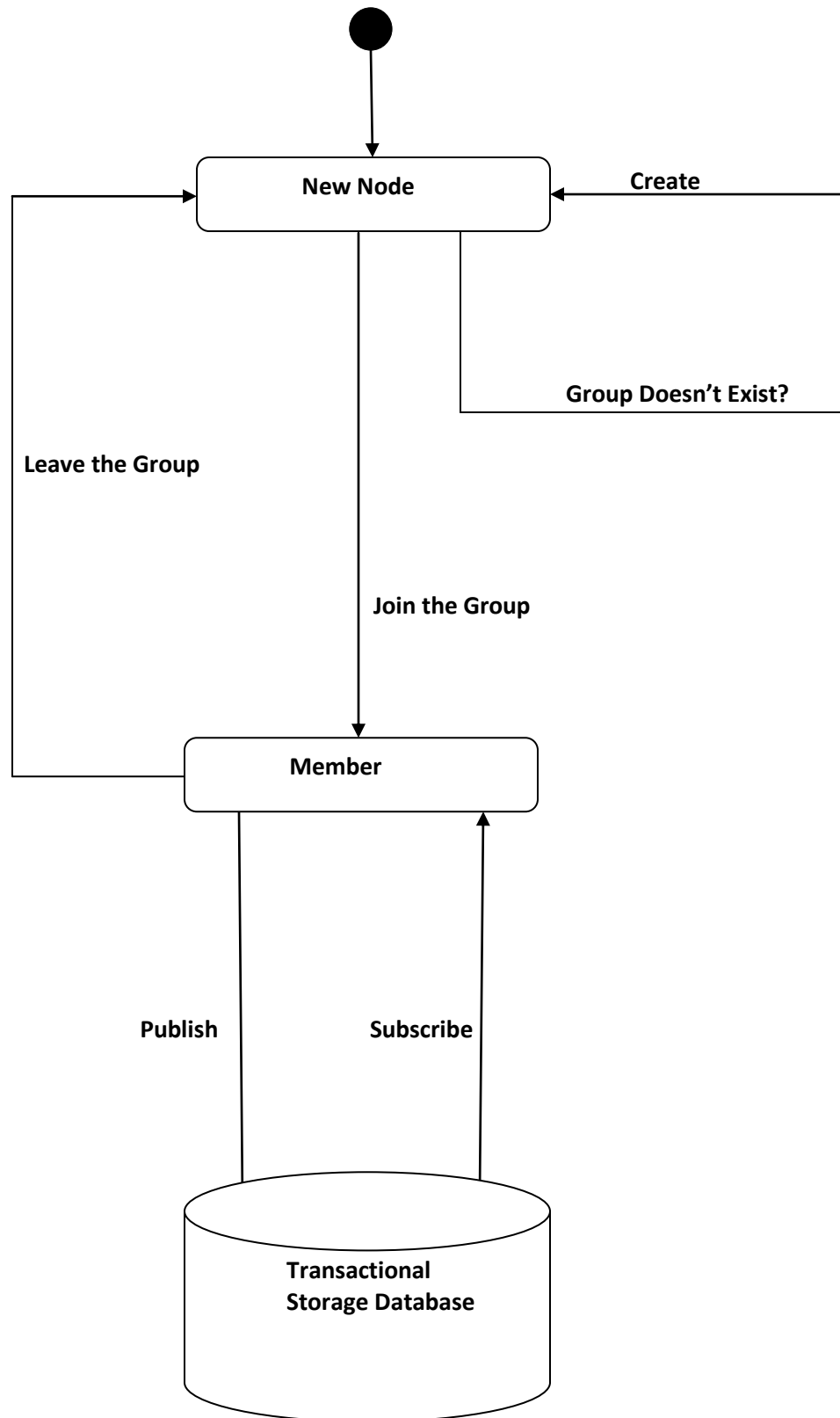


Figure 4.4 A state diagram displaying the nodes and its activities in a group communication system

5. Algorithms Used in Implementation of GComm

5.1 Join Group

The complete infrastructure of publish-subscribe implemented on the scalaris database is explained in the previous algorithms. The publisher will publish the topic name with a publisher ID, the value corresponding with a sequence number is written on the topic name. The subscriber will subscribe to the topic with subscriber ID and a definite amount of time period called the lease. The subscriber once gets subscribed can read the value corresponding to a sequence number provided the subscriber is in lease. The publisher/subscriber can delete the value corresponding to the sequence number on the topic published, and if the lease is expired the value gets automatically deleted.

This algorithm explains the procedure of the assign the group to the scalaris database. There are additional parameters like members list, readers list, and makers list. And there will be time parameters, time out and delay which are defined in the configuration file of the group communication. The group name after written into the scalaris has two states, stable and unstable. The group entry record is stable if the number of readers in readers list is equal to the number of the members in the members list. There is a view which is to be updated when there is group entry record written to the scalaris. And whenever there is group entry record, there is a list of parameters which will be written in group entry record, they are group name, node name, members list, readers list, makers list and the status(stable/unstable).

The inputs for the Join group algorithm are the configuration file of group communication. After getting the configuration file, the group name and node name are

retrieved from it. If group name does not exist in scalaris, then the list of the parameters mentioned above will be written to the scalaris. And if group name entry record is already written to scalaris, and it is unstable then the time out and delay values are retrieved from the configuration file. After retrieving them, if the timeout value is greater than zero and the group name entry is unstable then the timeout value is decremented by the value of one and there will be a wait time of delay value. The above procedure is a loop; the loop is executed until the timeout value becomes zero. When the time out value will become zero then the group record entry is written to scalaris, then the list of the parameters are written to the scalaris and the group record entry is updated as unstable. Then self view is updated

Algorithm:

Inputs

The configurations file of group Communication

groupName <= Retrieve group name from the configurations file

nodeName <= Retrieve node name from the configurations file

If *groupName* does not exists in Scalaris **then**

Write to Scalaris <*groupName*, {*nodeName*} as *MembersList*, { } as *ReadersList*,

{*nodeName*} as *LastChangemakerslist*, *stable*>

Update view

Else if *groupName* entry Record is Unstable **then**

timeOut <= Retrieve Time-Out value from configurations file

delay <= Retrieve the delay time from configurations file

while *timeOut* > 0 ^ *groupName* entry Record is Unstable **do**

timeOut <= *timeOut* - 1

wait(*delay*)

End while

Write to Scalaris $\langle \text{groupName}, \{\text{nodeName}\} \cup \{\text{MembersList}\}, \{\}, \{\text{nodeName}\}$
 $\cup [\text{LastChangemakerslist}], \text{unstable}\rangle$
Update self view

End if

5.2 Read Group State

In the previous algorithm, the group record entry is written into the scalaris, with a list of parameters like Members list, Readers list, change makers list and the status of the group entry record (stable/unstable).

In the read group state, the group entry record is retrieved from the scalaris. If it is stable then the state of the group which is stable is returned, else entry record is unstable. For all change maker node belongs to the members list and it does not belong to the self node name, and then execute the following loop.

If the change maker node belongs to members list and it does not belong to view then add the change maker node to the view. And if the change maker node does not belong to members list and it belongs to view then remove the change maker node from the view. Then add the self node to the readers list. If the number of readers in the readers list equals to the number of the members in the members list then make the group entry record stable otherwise make it unstable.

Algorithm

Retrieve the group entry record from scalaris $\langle \text{groupName}, \text{MemberList}, \text{ReadersList}, \text{ChangeMakersList}, \{\text{stable/unstable}\} \text{Flag} \rangle$

If Group has state *stable* **then**

```

    Return
End If
For all (changeMakerNode  $\in$  ChangeMakersList)  $\wedge$  (changeMakerNode  $\neq$  selfNodeName)
Do
    If (changeMakerNode  $\in$  MembersList)  $\wedge$  (changeMakerNode  $\notin$  view) then
        Add changeMakerNode to view
    Else If (changeMakerNode  $\notin$  MembersList)  $\wedge$  (changeMakerNode  $\in$  view) Then
        Remove changeMakerNode from view
    End if
End for
Add selfNode to ReadersList
If Number of Readers Equal Number of Members then
    Make Group Entry Record stable
Else
    Make Group Entry Record unstable
End If

```

5.3 Partition Algorithm

A partition occurs between the groups when there is a failure of network resulting in communication gap between the nodes in a group. When GComm detects the network failure, the node implements the partition algorithm. In this algorithm, when there is a split in the group due to network failure, the nodes form a subgroup and begin to function individually in the subgroups. The subgroups have the same group name as of the original group name, and they differ by the number of nodes and their names in them. When the network connection resume, the subgroups containing the nodes form together to the original group with the same group name. In this way our group communication system implements one of the objectives of CAP theorem, partition tolerance.

For this algorithm, the configuration file of group communication system along with group entry record containing group name, node name, members list, readers list and change makers list are retrieved. If partition occurs, the members list and the readers list are made empty making group entry stable. Then the node will add itself to the members list first, and then to the readers list making the group entry stable. Similar procedure is followed in the merger of two sub groups resulting after resuming the network connection.

Algorithm

Retrieve the group entry record from scalaris < *GroupName*, *MemberList*, *ReadersList*, *ChangeMakersList*, {*stable/unstable*} *Flag* >

If *partition* occurs then

- Make *MemberList* empty
- Make *Readerslist* empty
- Make group entry record *unstable*
- Add itself to *Memberslist*
- Add itself to *Readerslist*
- Add itself to *ChangeMakersList*
- Make group entry record *unstable*

Else If *merger* occurs then

- Make *MemberList* empty
- Make *Readerslist* empty
- Make group entry record *unstable*
- Add itself to *Memberslist*
- Add itself to *Readerslist*
- Add itself to *ChangeMakersList*
- Make group entry record *unstable*

End If

5.4 Register and Publish

A topic name which is to be published has to get registered in the scalaris database. Values will be written on that particular topic. Initially the topic should get registered with a unique name. The inputs of the register and publish algorithm is the topic name of data type string. If that particular topic name is not yet registered, then the output of this algorithm will be an ID which is randomly generated integer. After the ID is generated, the topic name and the randomly generated ID will be written to the scalaris database. The ID is a publisher ID, on which topic name gets published.

Algorithm:

Inputs

String : *topicName*

Outputs

Integer : *Publisher ID*

Require: A unique String *topicName*

Ensure: A new positive integer *ID*

If *topicName* exists **then**

Return -1

Else

ID <= Randomly Generated Positive Integer

Write to Scalaris <*topicName*, *ID*, -1>

Return *ID*

End If.

5.5 Write a Value on a Topic

After generating a randomly generated positive integer ID with the unique topic name, the value has to be written on that particular topic. The value can be anything string or integer. The

inputs of this particular algorithm are topic name, publisher ID associated with the topic name and the value which has to be written on the topic. The output will be the Sequence number, associated with the value.

The publisher ID has to publish the topic name in the scalaris database, then the value will get associated with the topic name. After the publisher ID writes a value on the topic name and a sequence number will be generated, and will be written on scalaris database along with the topic name and the publisher ID. When the sequence number is written to the scalaris database, the Global Virtual Time (GVT) will be retrieved at the time of writing to the scalaris. The Global Virtual Time will also be written to the scalaris, associated with the sequence number, topic name and publisher ID.

Algorithm:

Inputs

String: *topicName*

Integer: *Publisher ID*

String: *Value*

Outputs

Integer: *SequenceNo*

Require: Any String Value and a *topicName* published by publisher *ID*

Ensure: The sequence number *SequenceNo* of the value published.

If *ID* publishes *topicName* then

Retrieve the entry $\langle \text{topicName}, ID, PrSeqNo \rangle$

$SequenceNo \leq PrSeqNo$

Write to Scalaris $\langle \text{topicName}, ID, SequenceNo \rangle$

$GVT \leftarrow \text{Get Global Virtual Time}$

Write to Scalaris $\langle \langle \text{topicName}, ID \rangle, SequenceNo, GVT \rangle$

Return *SequenceNo*

End if

5.6 Publish a Topic

After writing the value on the topic name, it has to be published in the scalaris database. We have publisher ID, topic name with value, the sequence number is associated with the value and the Global Virtual Time (GVT) available. In order to publish the topic name on the scalaris database, we need the topic name and the publisher ID, so that through the ID , the publisher of the topic name can be traced. The inputs of this particular algorithm are topic name and publisher ID, and the output will be the Boolean status, which will ensure whether the topic name is published or not. If there is a name conflict in the topic name, then this algorithm will not allow publishing the topic name with the same name of the topic name which is already published.

Algorithm:

Inputs

String: *topicName*

Integer: *Publisher ID*

Outputs

Boolean: *Status*

Require: A unique string *topicName* and a registered publisher ID

If Topic *topicName* is not published yet **then**

Write to Scalaris <*topicName*, *ID*, -1>

Return true

Else

Return false

End if

5.7 Subscribe to a Topic

We have topic name and its publisher ID has published in the scalaris database. If this topic name has to be accessed, then it needs to be subscribed. The users who subscribes to the topic published are called the subscribers. The subscription to the topic will be given for only a certain amount of the time called lease time. The lease time has to be mentioned at the point of subscription. Similar to the publisher, the subscriber will be assigned a randomly generated positive ID called Subscriber ID. In other words, the subscriber will subscribe to the topic published with parameters lease time and the Subscriber ID. For each topic, there can be none or more subscribers. If the topic has one or more subscribers, then the details of subscribers of that topic can be retrieved by the option Subscriber list. And whenever the subscriber will subscribe to the topic, it will get added to the list of subscribers of that topic. The list of subscribers of a topic will have the details of the subscribers i.e., the list will have the subscribers ID and their lease time.

The inputs of this algorithm will be the topic name and the parameter lease time, and the output will be the ID, in other words subscriber ID. The lease should be a positive integer, the subscriber ID generated will be a random positive integer. If the topic name is already published, then the subscriber ID will be added to the list of the subscribers with the lease time associated with it. And if the topic name is not yet published then, the subscriber will be added to the list of the subscribers of the topic with the lease time associated with it. As the topic name is not yet published, the topic name and the list of the subscribers of that topic will be written on the scalaris database.

Algorithm

Inputs

String : *topicName*

Integer : *Lease*

Outputs

Integer : *ID*

Require: *Lease* >= 0

ID <= Randomly Generated Positive Integer

If Topic *topicName* is already published then

Add to the list of subscribers <ID, *Lease*, -1>

Else

SubscriberList <= Get list of Subscribers of *topicName*

SubscriberList := [<ID, *Lease*, -1>]

Write to Scalaris <topicName, NULL, -1, *SubscriberList*>

End If

Return *ID*

5.8 Subscribe to a Topic with an ID

In earlier algorithm, subscriber ID is created when it initially tries to subscribe to a topic with a certain amount of the lease time. In this way, the subscriber will get an ID for itself. A subscriber can subscribe to one or more topics with same or different lease times. So the subscriber with a subscriber ID can subscribe to one or more topics with different lease times. The subscriber with an ID can resume the accessibility to the topic at the same point if he exits at that point of accessing the topic provided the subscription is valid i.e., the lease time shouldn't expire. This will ensure the continuity of the accessing to the topic by the subscriber with an ID. For every subscription to the topic, there will be parameters *LastState* and *LastReadSequenceNumber*. The *LastState* refers to the last state of the topic name, it will

mention whether the topic is still published or not. The *LastReadSequenceNumber* will be the last sequence number of the topic name read by subscriber ID.

In this particular algorithm, the inputs will be a topic name, Subscriber ID and lease. The output will be *LastState* of the topic name and the *LastReadSequenceNumber* of the topic name read by the subscriber ID. The lease time should be a positive integer. If subscriber ID is not subscribed to a topic name then add the subscriber ID to the subscribers list of the topic name. If subscriber ID is already subscribed to the topic, then this algorithm will return the *LastState* of the topic name and the *LastReadSequenceNumber* of the topic name read by the subscriber ID.

Algorithm:

Inputs

String : *topicName*

Integer : *ID*

Integer : *Lease*

Outputs

String, Integer : *LastState*, *LastReadSequenceNo*

Require: Lease ≥ 0

If *ID* is not subscribed to *topicName* then

 Add *ID* to subscribers list of *topicName*

Else

LastReadSequenceNo \leftarrow Get last sequence number of *topicName* read by
subscriber ID

LastState \leftarrow Get Last state of *topicName*

Return \langle *LastState*, *LastReadSequenceNo* \rangle

End if

5.9 Read Value

Earlier Algorithms mentioned the procedure of subscriptions to the topic. Every topic has a publisher ID and every Subscriber has a unique subscriber ID. The ultimate reason behind the publish-subscribe procedure is subscriber should read the value/values in the topic published by the publisher, provided the subscriber is subscribed to that topic. And every topic has sequence number which is last read by the subscriber with an ID, through which it can retrieve the value when it tries to read the topic again.

In this algorithm, in order to read a value published on a topic by a publisher we need topic name and the subscriber ID as inputs, then we get value, sequence number which is last read by the subscriber ID and the sequence number will be incremented by the value of one. After getting topic name and the subscriber ID as inputs, then the algorithm checks if there is any value corresponding to the sequence number exists on the given topic name. If there is value associated with the sequence number, it will retrieve the time stamp of the sequence number on which value is written on the topic name, Global Virtual Time (GVT) and the lease time parameter which was mentioned at the time of subscription to a topic published. If the Global Virtual Time (GVT) is greater than the sum of the time stamp and the lease parameter then the sequence number is updated with the increment by the value of one and returns the value published on that topic at that sequence number. If the Global Virtual Time (GVT) is less than the sum of the time stamp and the lease parameter then the last sequence number is updated as the current sequence number and the value published on the topic corresponding to the topic is returned. And if there is no value corresponding to the sequence number which exists on the topic name then the maximum sequence number retrieved. The maximum

sequence number corresponds to the sequence number of the last written value on the topic name. If maximum sequence number is greater than or equal to the current sequence number, the value has been deleted by the garbage collector and the NULL value is returned. And if maximum sequence number is less than the current sequence number, then the sequence number seeking is not yet published, the same sequence number is maintained and will execute the loop again. The above procedure mentioned in this algorithm is a loop process. The loop is executed till the end until the lease period gets expired.

This is the regular procedure followed in order to read the value corresponding to a sequence number published under a topic name by the subscriber who subscribed to that particular topic with a subscriber ID.

Algorithm:

Inputs

String: *topicName*

Integer: *ID*

Outputs

String : *Value*

SeqNo <= Retrieve the sequence number last read by *ID*

SeqNo <= *SeqNo* + 1

Loop

If Value corresponding to *SeqNo* exists on *topicName* **then**

TimeStamp <= Retrieve time Stamp of *SeqNo*th Value written on Topic
topicName

GV T <= Get Global Virtual Time

Lease <= Get Lease of Subscriber ID on *topicName*

if *GV T* > *TimeStamp* + *Lease* **then**

```

        SeqNo <= SeqNo + 1
    Else
        Update Last Read Sequence number as SeqNo
    Return Value corresponding to SeqNo
End if
Else
    MaxSeqNo <= Retrieve Sequence number of the last written Value on
    Topic topicName
    If MaxSeqNo >= SeqNo then
        Return NULL as the Value has been deleted by Garbage collector
    Else
        SeqNo is not yet published hence maintain the same SeqNo and
        loop to wait.
    End if
End if
End loop

```

5.10 Read Sequence Number's Value

In Previous algorithm, we have seen the procedure of reading the value corresponding to a sequence number published under a topic name by the subscriber who subscribed to that particular topic with a subscriber ID. In the read sequence number's value algorithm, we explain how to read a value corresponding to a particular sequence number published on a topic by a publisher ID provided the subscriber is subscribed to the topic name. The subscriber should subscribe to the topic with a unique subscriber ID and the current Global Virtual Time (GVT) should be greater than the sum of the time stamp of the value corresponding to that sequence number of the topic published and the lease time period which was mentioned at the point of subscription to the topic.

In this algorithm, to read the value corresponding to a particular sequence number given we should have topic name, subscriber ID and sequence number as the inputs, then we can retrieve the value corresponding to that particular sequence number can be retrieved. The procedure followed is almost similar to the previous algorithm i.e., read the value algorithm, the only difference being the input. In this algorithm, we give sequence number as input parameter, while in the previous algorithm we don't have the sequence number as the input. After getting topic name, the subscriber ID and the sequence number as input, then the algorithm checks if there is any value corresponding to the sequence number exists on the given topic name. If there is value associated with the sequence number, it will retrieve the time stamp of the sequence number on which value is written on the topic name, Global Virtual Time (GVT) and the lease time parameter which was mentioned at the time of subscription to a topic published. If the Global Virtual Time(GVT) is greater than the sum of the time stamp and the lease parameter then the sequence number is updated with the increment by the value of one and returns the value published on that topic at that sequence number. If the Global Virtual Time(GVT) is less than the sum of the time stamp and the lease parameter then the last sequence number is updated as the current sequence number and the value published on the topic corresponding to the topic is returned. And if there is no value corresponding to the sequence number which exists on the topic name then the maximum sequence number retrieved. The maximum sequence number corresponds to the sequence number of the last written value on the topic name. If maximum sequence number is greater than or equal to the current sequence number, the value has been deleted by the garbage collector and the NULL value is returned. And if maximum sequence number is less than the current sequence

number, then the sequence number seeking is not yet published, the same sequence number is maintained and will execute the loop again. The above procedure mentioned in this algorithm is a loop process. The loop is executed till the end until the lease period gets expired.

Algorithm:

Inputs

String: *topicName*

Integer: *ID*

Integer: *SeqNo*

Outputs

String : *Value*

SeqNo <= Retrieve the sequence number last read by *ID*

SeqNo <= *SeqNo* + 1

Loop

If Value corresponding to *SeqNo* exists on *topicName* **then**

TimeStamp <= Retrieve time Stamp of *SeqNo*th Value written on Topic
topicName

GV T <= Get Global Virtual Time

Lease <= Get Lease of Subscriber ID on *topicName*

If *GV T* > *TimeStamp* + *Lease* **then**

SeqNo <= *SeqNo* + 1

Else

Update Last Read Sequence number as *SeqNo*

Return Value corresponding to *SeqNo*

End if

Else

MaxSeqNo <= Retrieve Sequence number of the last written
Value on Topic *topicName*

```

    If MaxSeqNo >= SeqNo then
        Return NULL as the Value has been deleted by Garbage collector
    Else
        SeqNo is not yet published hence maintain the same SeqNo and loop to
        wait.
    End if
End if
End loop

```

5.11 Delete the Sequence Number's Value

In earlier algorithms, we've seen the procedure of subscription to a topic published and how to read the value corresponding to the sequence number on a topic published. The publisher publishes a topic with a publisher ID and the subscriber subscribes to the topic published with the subscriber ID and lease period. Every subscriber can subscribe to one or more topics with a corresponding unique subscriber ID. And the subscriber can read the value on the basis of sequence number and resume the subscription with the last sequence number read by the subscriber.

In this delete the sequence number's value algorithm, the procedure of deleting the value corresponding to the sequence number will be explained. The value deleted will be collected by the garbage collector. In order to delete the value corresponding to the sequence number, we need topic name, publisher ID and sequence number as input. Then the value corresponding to the sequence number is deleted which is the output. After we get the inputs required, then the algorithm checks whether the value corresponding to the sequence number exists on the topic name or not. If exists, then the publisher ID will be added to the delete-

doers list. The delete doers list consists of the pair topic name and sequence number. And if there is no value corresponding to the sequence number, then the algorithm is not executed.

Algorithm:

Inputs

String: *topicName*

Integer: *Publisher ID*

Integer: *SeqNo*

If Value corresponding to *SeqNo* exists on *topicName* **then**

 Add ID to Delete-Doers list of <*topicName*, *SeqNo*>

End if

5.12 Delete the Sequence Number's Value with the State

In the previous algorithm, we have seen the procedure of deleting a value corresponding to a particular sequence number of a topic published by the publisher ID. In the delete the sequence number's value with the state algorithm, we follow the similar procedure of the delete the sequence number's value algorithm, and additionally we update the state of the publisher ID to the *State*. Every Publisher ID will have a state which mentions whether there are any topics which are publishing or stopped publishing.

In this delete the sequence number's value with state algorithm, the procedure of deleting the value corresponding to the sequence number and updating the state of the subscriber will be explained. The value deleted will be collected by the garbage collector. In order to delete the value corresponding to the sequence number, we need topic name, subscriber ID, sequence number and state of subscriber as inputs. Then the value

corresponding to the sequence number is deleted which is the output. After we get the inputs required, first it will update the state of the subscriber ID to *the state* then the algorithm checks whether the value corresponding to the sequence number exists on the topic name or not. If exists, then the publisher ID will be added to the delete-doers list. The delete doers list consists of the pair topic name and sequence number.

Algorithm:

Inputs

String: *topicName*

Integer: *Subscriber ID*

Integer: *SeqNo*

String: *State*

Update state of *ID* to *State*

If Value corresponding to *SeqNo* exists on *topicName* **then**

Add ID to Delete-Doers list of <*topicName*, *SeqNo*>

End if

5.13 Leave Group

In previous two algorithms, the publishers and subscribers join the group, then written into scalaris. The group record entry state can also be retrieved, whether it is stable or unstable. If the publishers and subscribers want to leave the group, then the leave group algorithm is implemented.

The inputs for the leave group algorithm are the configuration file of group communication. After getting the configuration file, the group entry record is retrieved from it. The group entry record consists of the list of parameters group name, members list, readers

list, change makers list and state of the group entry record (stable/unstable). Also the time based parameters time out and delay values are retrieved from the configuration file. After retrieving them, if the time out value is greater than zero and the group name entry is unstable then the time out value is decremented by the value of one and there will be a wait time of delay value. The above procedure is a loop which gets executed until the group name entry record is stable. When the time out value will become zero then the group record entry is written to scalaris with the list of the parameters group name, members list, self node, readers list, change makers list and make the group record entry as unstable. Thus the publishers and subscribers leave the group.

Algorithm

Inputs

The configuration file of group Communication.

Retrieve the Group Entry Record from Scalaris *<groupName, MembersList, ReadersList, ChangeMakersList, {stable/unstable}Flag>*

Timeout <= Retrieve Time-Out value from Configuration file

Delay <= Retrieve the delay time from Configuration file

While *timeOut* > 0 ^ *groupName* entry Record is Unstable **do**

Timeout <= *timeOut* - 1

Wait(delay)

End while

Write to Scalaris *<groupName, MembersList \ {selfNode}, { }asReadersList, ChangeMakersList U {selfNode}; unstable>*

5.14 Garbage Collector

In the previous two algorithms, the procedure of deleting a value corresponding to a sequence number of the topic name published by updating the state of the publisher ID is explained. There are three reasons to delete a value corresponding to the sequence number, they are as follows:

- The value can be deleted by the subscriber ID
- If the lease period of the subscription is expired

In the Garbage collector algorithm, the procedure of the deleting the value is explained. If the publisher/subscriber wish to delete or the lease period of the subscription is expired, then the value corresponding to the sequence number of the topic name published is deleted. The lease is expired, when the Global Virtual Time (GVT) is greater than the sum of the time stamp recorded at the point of publishing and the lease period value which is mentioned at the point of the subscription.

Algorithm:

```

For all topicName ∈ Topics published till now do
    For all seqNo ∈ Sequence numbers of values written on topicName do
        For all subscriber ∈ Subscribers of topicName do
            If subscriber has issued a delete ∨ lease of subscriber has expired then
                Delete < topicName, seqNo, Value >
            End if
        End for
    End for
End for

```

6. Related Work

We have compared different distributed databases and group communication systems with respect to group communication attributes like consistency, availability, partition tolerance, mutable objects, churn tolerance and real time response.

The OMG Data-Distribution Service (DDS) [9] provides the implementation of publish-subscribe concept, in which publisher publishes the message and subscriber subscribes to the messages published. The Data Distribution Service (DDS) have two layers of interfaces: the lower level of interface Data-Centric Publish Subscribe (DCPS) make sure that every message published will be delivered to the subscriber, and the higher level of interface Data-Local Reconstruction Layer (DLRL) enables the integration into the application layer. The DDS infrastructure doesn't provide all the group communication characteristics like consistency, availability, partition tolerance, mutable objects and churn tolerance. It provides only real time response. Some reliable multicast group communication systems like ISIS [5][6] and Spread[8] will provide only two of the group communication characteristics, partition tolerance and real time response. These systems are centralized and have single point failure, when there is failure in the node or component then the whole system fails.

Google's Bigtable [11], which is implemented in many Google applications like Google earth, Google Finance and web services is a distributed storage system for organizing the structured data. Bigtable has achieved the group communication characteristics like consistency, availability and partition tolerance; but they fail to achieve the other group communication characteristics churn tolerance, mutable objects and real time response.

Cassandra [12] is developed by the Facebook, is a distributed storage system managing the large data deployed on the servers located at various geographical locations. It runs on the top of the nodes located in various locations. It is decentralized and doesn't have single point failure. It achieves the group communication characteristics like availability and mutable objects, but fail to achieve other attributes consistency, partition tolerance, churn tolerance and real time response. Another database, Apache's CouchDB [13] is an open source documented oriented database. It is written in Erlang language, and it stores the data in the form of a collection of JSON documents instead of tables. The couchDB achieves group communication objectives, consistency and mutable objects. But it fails to achieve availability, partition tolerance, churn tolerance and lacks real time response.

Distributed Hash Table (DHT) [14] implements the key-based routing in a structured way, thus achieving the decentralization, fault tolerance and scalability. Also, it achieves the group communication characteristics like mutable objects and real time response. But it fails to implement Brewer's CAP theorem completely. It fails to achieve the group communication characteristics like consistency, availability, partition tolerance and churn tolerance. Amazon's dynamo is a key-value store [15]. To achieve the high availability, it gave up the consistency objective. It achieves availability, partition tolerance and mutable objects; but fails to achieve other group communication characteristics like consistency, churn tolerance and real time response. Last middleware system we compared is gizzard, which is implemented on twitter [16]. It runs on Java virtual machine (JVM) and achieves the objectives availability, partition tolerance, churn tolerance and real time response. It fails to implement the objectives of group communication system, consistency and mutable objects.

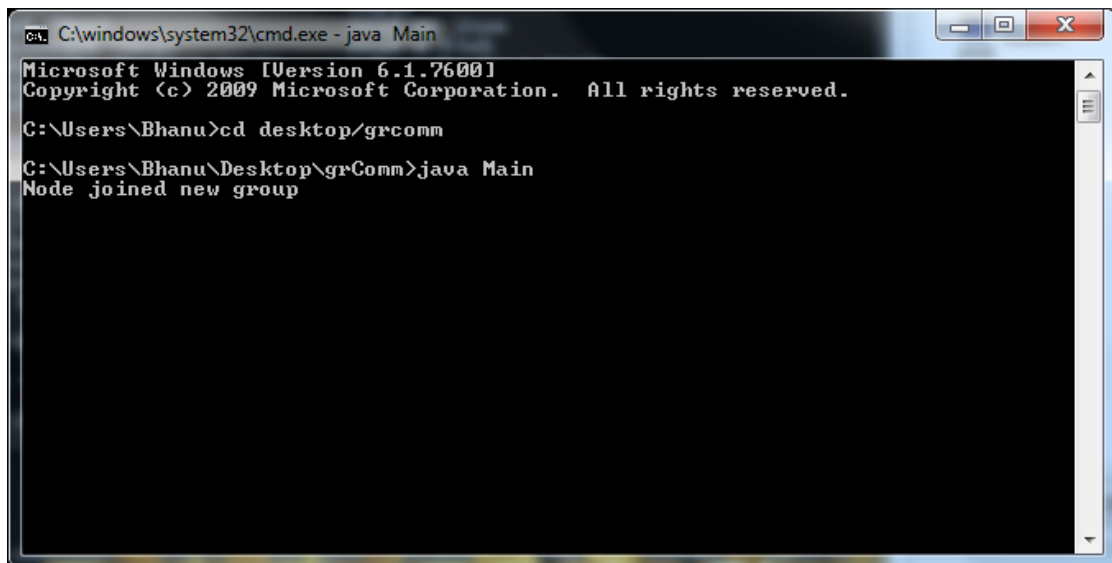
Unlike above group communication systems and distributed databases, our work reliable group communication system over asynchronous substrate implements all the group communication objectives with a slight modification in objective consistency. We achieved “Eventual Consistency” instead of consistency, availability, partition tolerance, churn tolerance, mutable objects and real time response. We designed and developed a reliable group communication system over an asynchronous substrate, the group communication system being GComm, and the asynchronous substrate being distributed database scalaris.

7. Experimental Results

To implement our design and development, we implemented our work on two nodes to get results and performance analysis. We've choose two nodes named Kayathi and Anvesh-PC. And we've configured them with scalaris database and GComm group communication system.

7.1 Screenshots

When two nodes: Kayathi and Anvesh-PC joins a group named root, the series of the events happened in explained in the screenshots below.



```
C:\windows\system32\cmd.exe - java Main
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Bhanu>cd desktop/grcomm
C:\Users\Bhanu\Desktop\grComm>java Main
Node joined new group
```

Figure 7.1.1 A screenshot of node kayathi joining the group

Initially, the node kayathi wants to join the group, root. The node admin of the kayathi will build the node configuration file with node name kayathi, group name node and IP address of the scalaris database. As the group name node doesn't exist in the group communication system, the system will create a group named root. In other words, as the group root doesn't exist, it is created by the node kayathi. After creating the group, it updates its view which is

shown below. The view shows the node kayathi is the one and only member of the group named root.

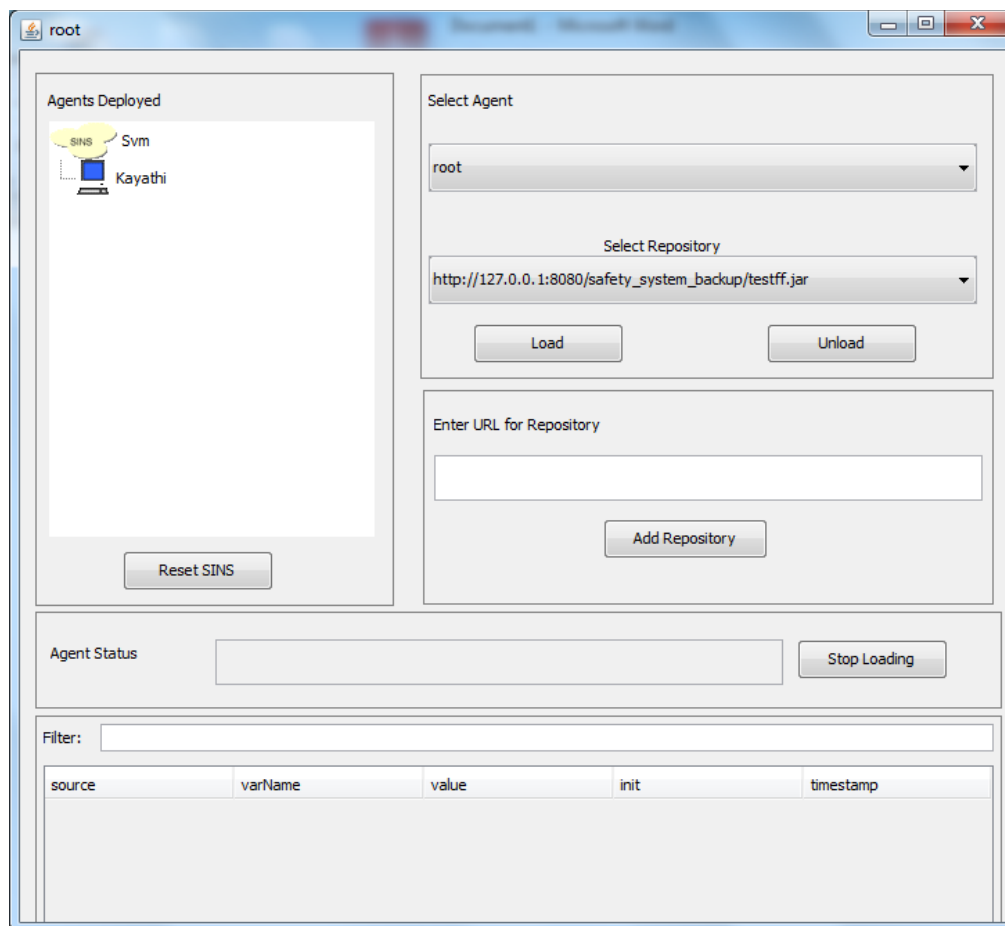
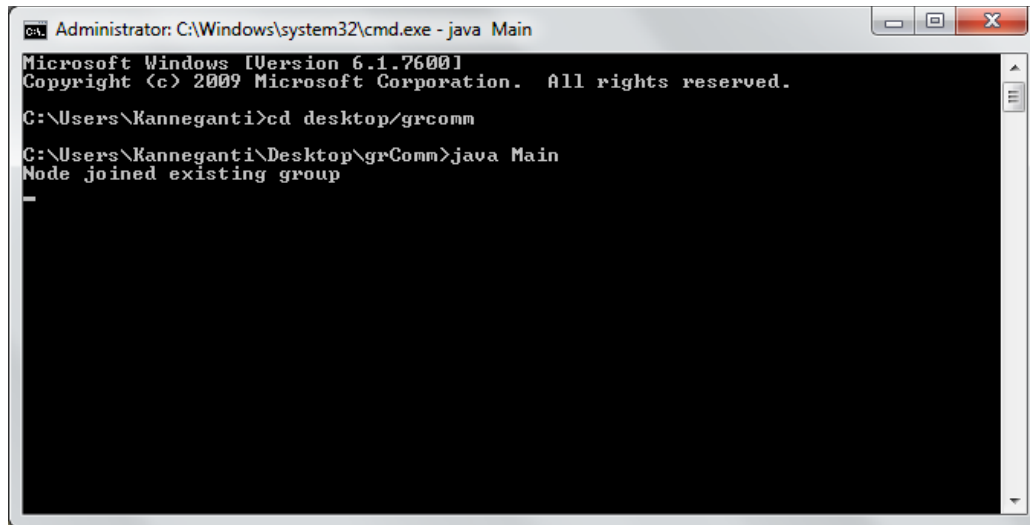


Figure 7.1.2 A screenshot of Node Kayathi's view after joining the group named root.

When the other node named anvesh-pc wants to join the group named root. The node administrator of the node anvesh-pc will build the node configuration file with node name anvesh-pc, group name root and IP address of the database. When the group communication system receives the request, it checks whether there is a group named root exists. As it was already created by the other node kayathi, it joins the node anvesh-pc into the group named root. Then the anvesh-pc updates its view, which shows both nodes kayathi and anvesh-pc are

members of group named root. The view will appear same to the both nodes kayathi and anvesh-pc.



```
Administrator: C:\Windows\system32\cmd.exe - java Main
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Kanneganti>cd desktop/grcomm
C:\Users\Kanneganti\Desktop\grComm>java Main
Node joined existing group
```

Figure 7.1.3 A screenshot of node Anvesh-pc joining the group

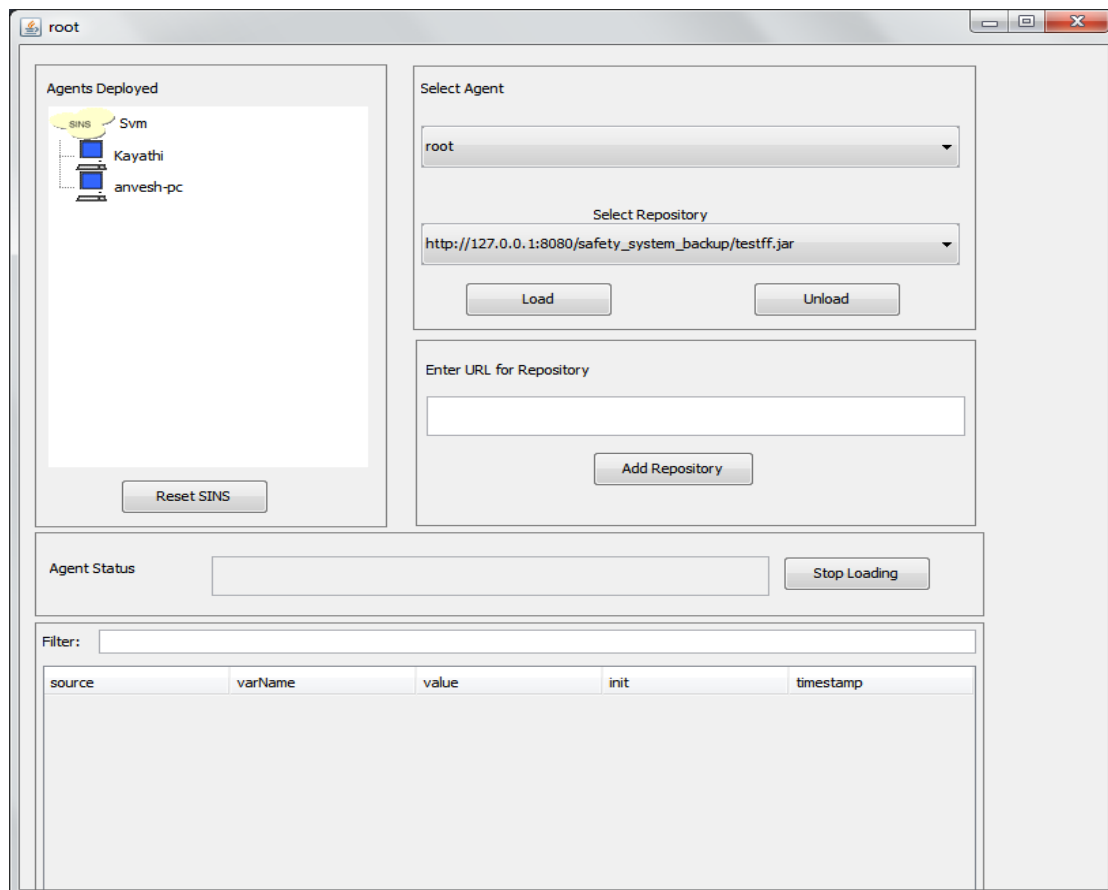
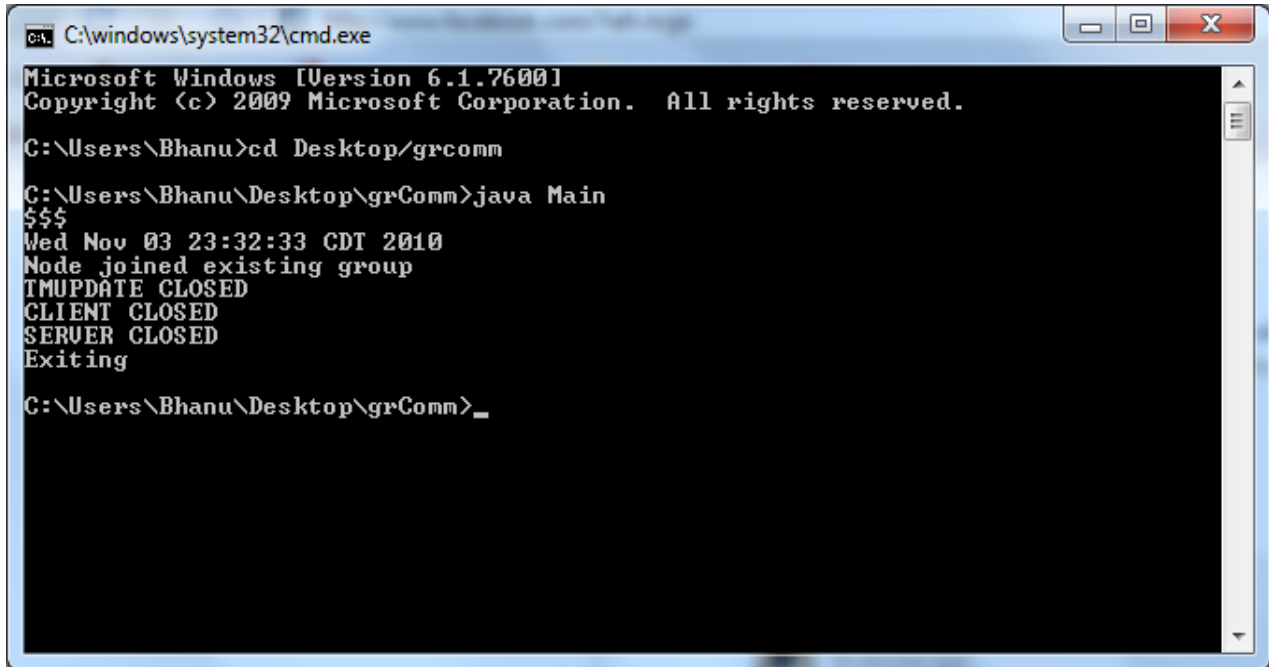


Figure 7.1.4 A screenshot of Node Anvesh-pc's View after joining the group named root.

The view showing two nodes, kayathi and anvesh-pc in the group named root.



```
C:\windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Bhanu>cd Desktop\grcomm
C:\Users\Bhanu\Desktop\grComm>java Main
$$$
Wed Nov 03 23:32:33 CDT 2010
Node joined existing group
TMUPDATE CLOSED
CLIENT CLOSED
SERVER CLOSED
Exiting
C:\Users\Bhanu\Desktop\grComm>_
```

Figure 7.1.5 A screenshot of terminal window, when node kayathi leaves the group.

When the node kayathi leaves the group named root, the terminal in the kayathi node displays the command “Exiting” as shown above. The view of node kayathi will be closed, as it’s not a member of any group in the group communication system. The node has to be a member of any group in the group communication system, if it is present in the system.

On the other hand, the node anvesh-pc is still the member of group named root, even though the creator of the group leaves the group. The view of node anvesh-pc will show only itself as the member of the group, root. And if the node anvesh-pc will leave the group root, then the group root will become nonexistent and the group communication system GComm will delete the group root. As mentioned earlier, a group in a GComm must contain at least one node as its member otherwise it will become nonexistent.

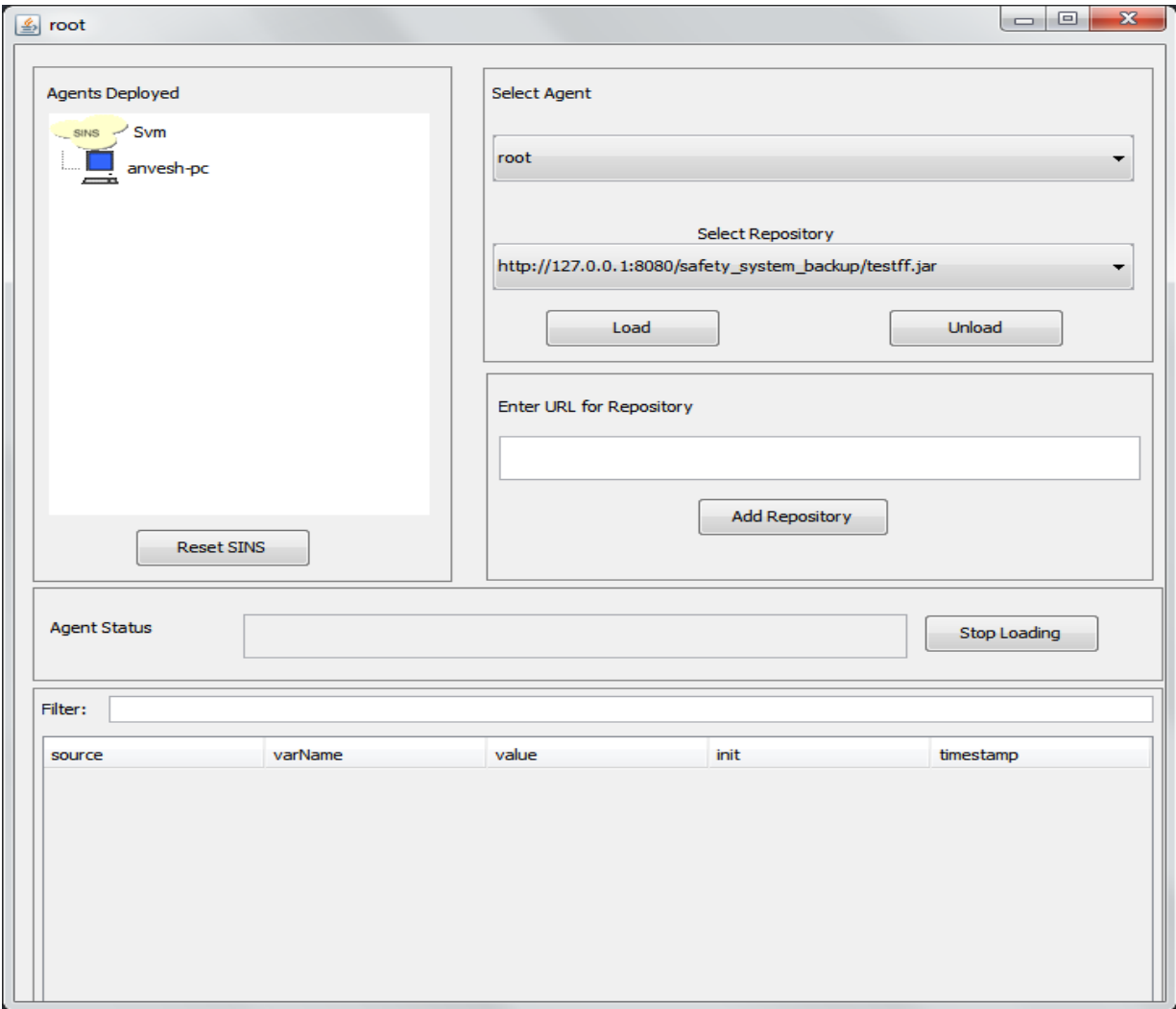


Figure 7.1.6 A screenshot of Node Anvesh-pc's view after the node kayathi leaves the group.

7.2 Performance Analysis

We recorded the time take for each activity carried by the node in GComm by running our execution for 30 iterations. The activities for which we recorded the time are node joining the group, leaving the group, writing a value, reading a value and deleting a value. The performance graphs (figure 7.2.1 and figure 7.2.2) has units of time in seconds, while the performance graphs (figure 7.2.3, figure 7.2.3 and figure 7.2.3) has units of time in milliseconds.

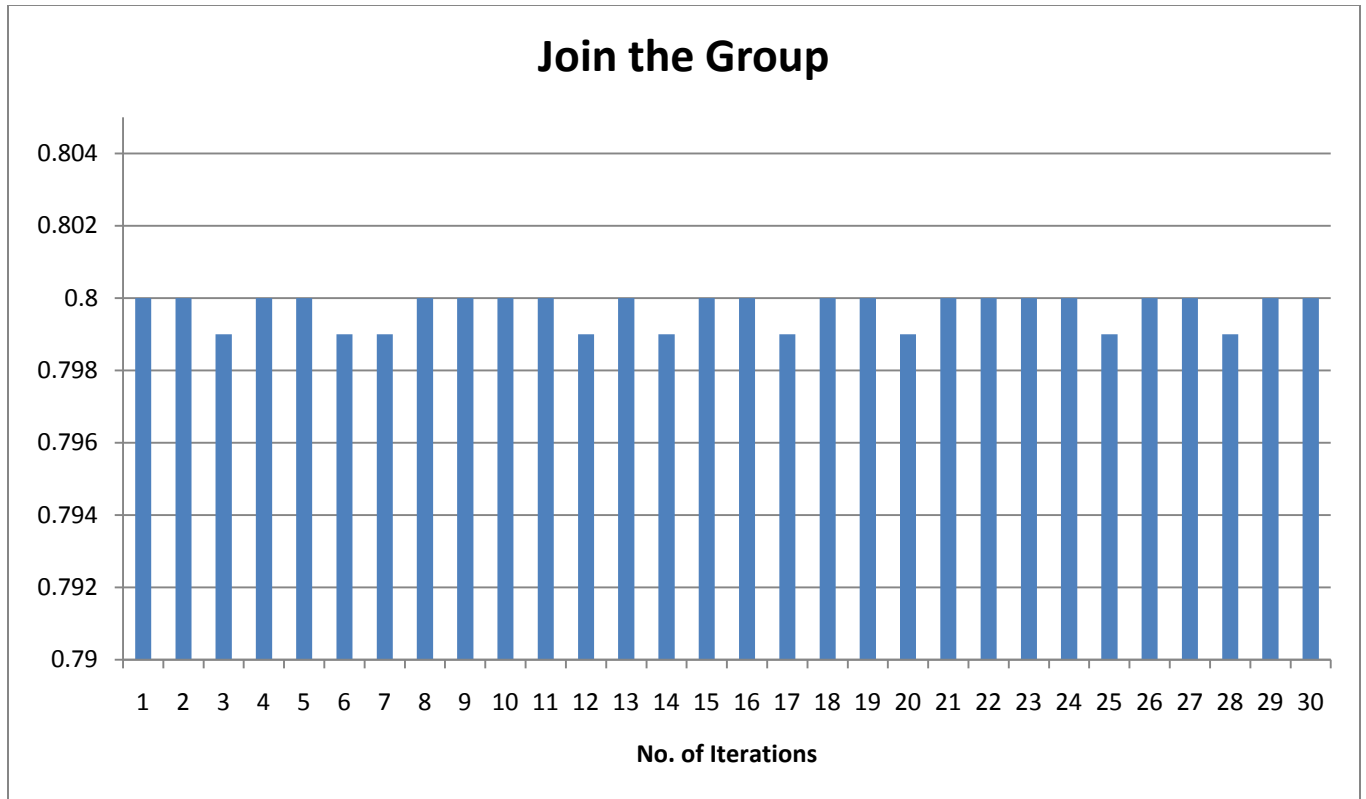


Figure 7.2.1 A performance graph displaying the time taken (in seconds) for node to join the group in 30 iterations

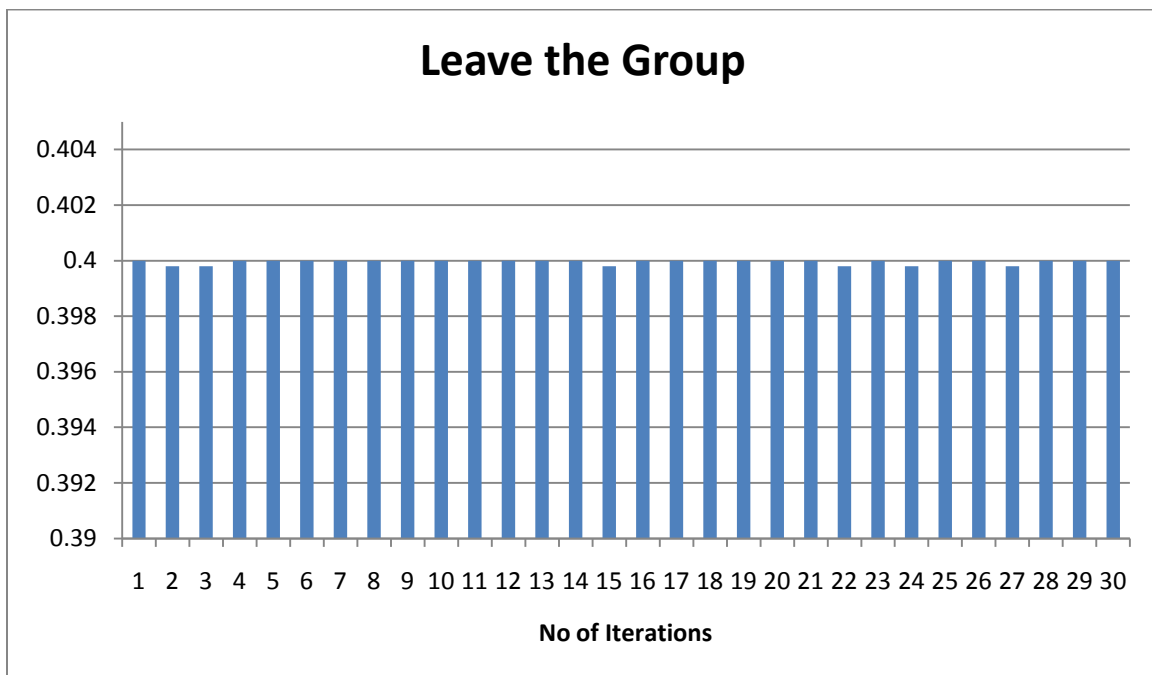


Figure 7.2.2 A performance graph displaying the time taken (in seconds) for node to leave the group in 30 iterations

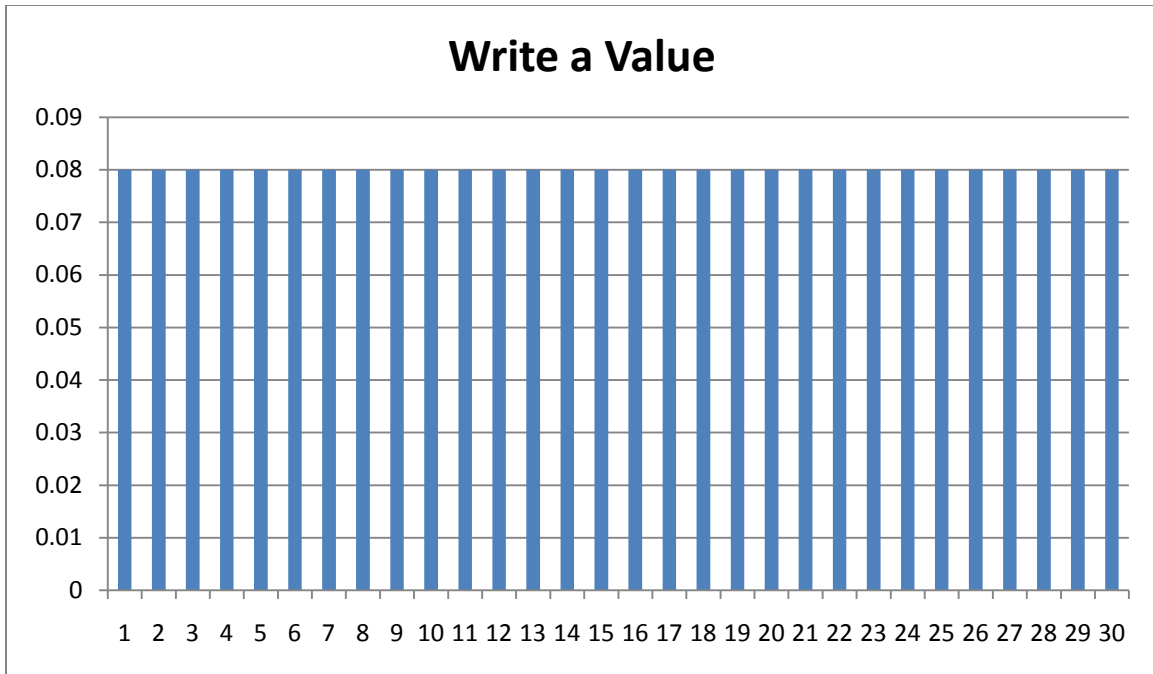


Figure 7.2.3 A performance graph displaying the time taken (in milliseconds) for writing a value

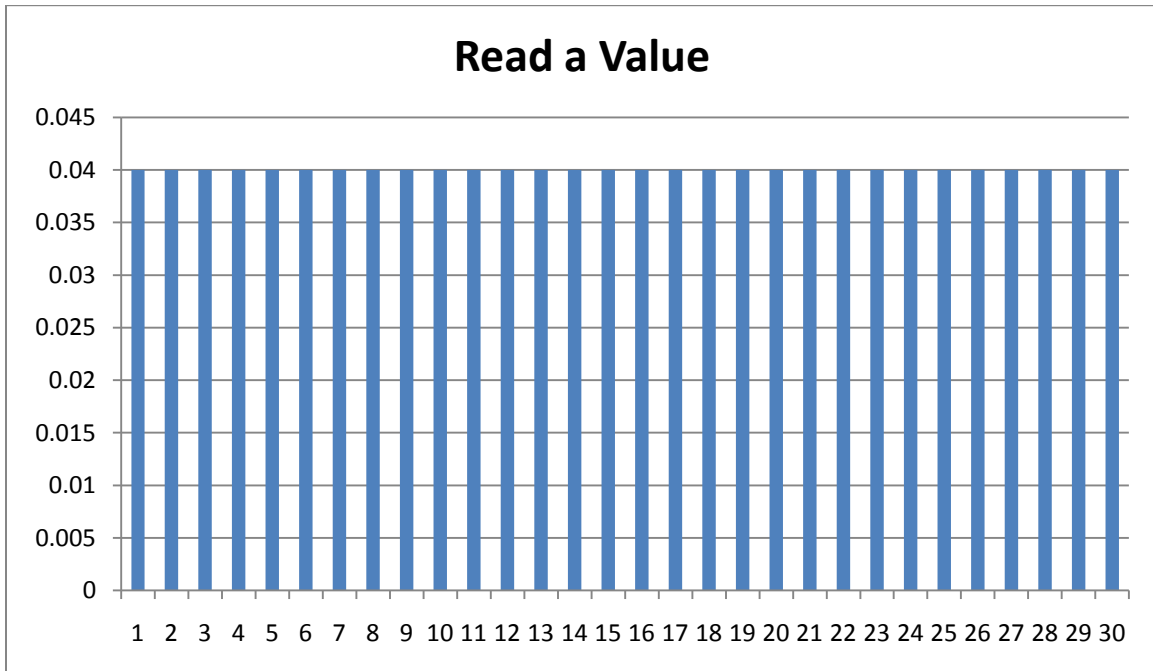


Figure 7.2.4 A performance graph displaying the time taken (in milliseconds) for reading a value

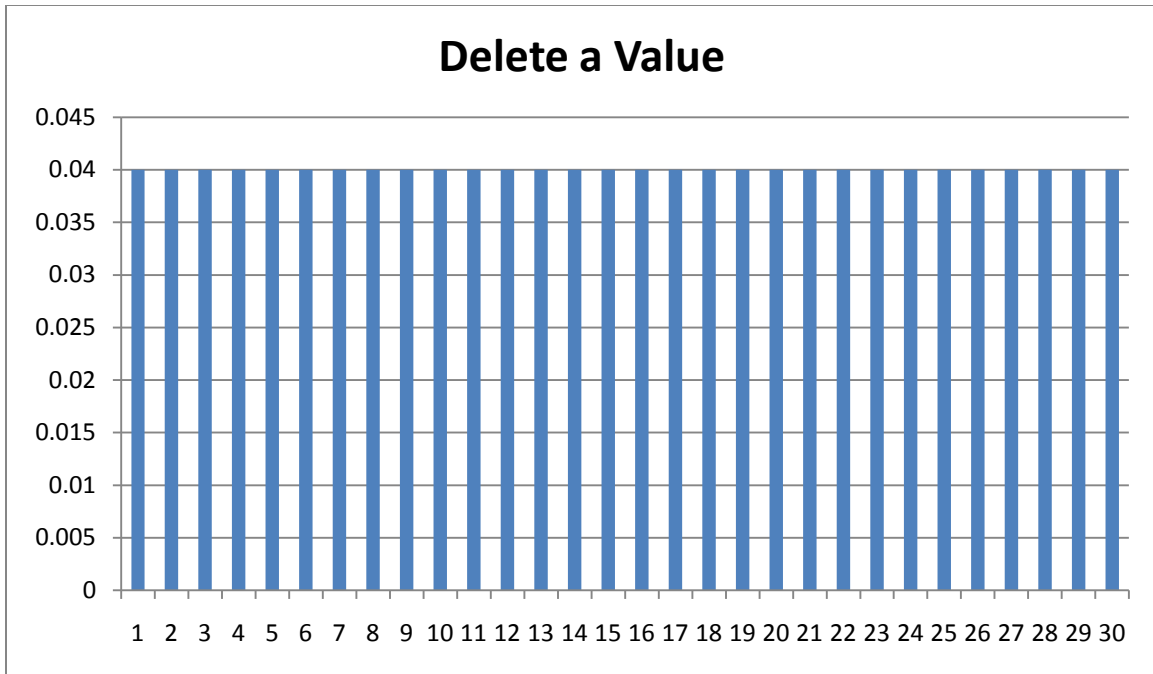


Figure 7.2.5 A performance graph displaying the time taken (in milliseconds) for deleting a value

8. Conclusions

We have designed and developed a reliable group communication system over an asynchronous substrate which provide persistent communication in disadvantaged networks and guarantees of

- Eventual Consistency
- Availability
- Partition Tolerance

Our system implemented publish-subscribe framework successfully. Our system comprises of mutable objects, which can be altered. Also, it achieved churn tolerance and real time response.

References

1. Lamport, L.; , "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," Computers, IEEE Transactions on , vol.C-28, no.9, pp.690-691, Sept. 1979 [doi: 10.1109/TC.1979.1675439]
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1675439&isnumber=35185>
2. W. Vogels. Eventually Consistent. ACM Queue vol. 6, no. 6, December 2008.
3. Stonebraker, Michael. "Errors in Database Systems, Eventual Consistency, and the CA P Theorem." Communications of the ACM. ACM, 05 04 2010. Web. 20 Oct 2010.
<<http://cacm.acm.org/blogs/blog-cacm/83396-errors-in-database-systems-eventual-consistency-and-the-cap-theorem/fulltext>>.
4. Eric A. Brewer, Towards robust distributed systems (abstract), Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing, p.7, July 16-19, 2000, Portland, Oregon, United States [doi>10.1145/343477.343502]
5. Kenneth P. Birman, Reliable Distributed Systems: Technologies, Web Services, and Applications, Springer-Verlag New York, Inc., Secaucus, NJ, 2005
6. Kenneth P. Birman. 1993. The process group approach to reliable distributed computing. *Commun. ACM* 36, 12 (December 1993), 37-53. DOI=10.1145/163298.163303
<http://doi.acm.org/10.1145/163298.163303>
7. B. Ban. JGroups, a toolkit for reliable multicast communication.
<http://www.jgroups.org/>, 2002.
8. AMIR,Y.AND STANTON, J. 1998. The spread wide area group communication system. TR CND5-98-4, The Center for Networking and Distributed Systems, The Johns Hopkins University.
9. Pardo-Castellote, G.; , "OMG Data-Distribution Service: architectural overview," Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on , vol., no., pp. 200- 206, 19-22 May 2003
doi: 10.1109/ICDCSW.2003.1203555
URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1203555&isnumber=27094>.
10. Thorsten Schett, Florian Schintke, and Alexander Reinefeld. 2008. Scalaris: reliable transactional p2p key/value store. In *Proceedings of the 7th ACM SIGPLAN workshop on ERLANG* (ERLANG '08). ACM, New York, NY, USA, 41-48. DOI=10.1145/1411273.1411280
<http://doi.acm.org/10.1145/1411273.1411280>
11. Fay Chang , Jeffrey Dean , Sanjay Ghemawat , Wilson C. Hsieh , Deborah A. Wallach , Mike Burrows , Tushar Chandra , Andrew Fikes , Robert E. Gruber, Bigtable: A

Distributed Storage System for Structured Data, ACM Transactions on Computer Systems (TOCS), v.26 n.2, p.1-26, June 2008 [doi>10.1145/1365815.1365816] Scalaris

12. A. Lakshman, P. Malik. Cassandra - A decentralized structured storage system, in: 3rd ACM SIGOPS International Workshop on Large Scale Distributed Systems and Middleware, 2009.
13. "The CouchDB Project." Apache CouchDb: The CouchDB Project. The Apache Software Foundation, 2009. Web. 20 Oct 2010. <<http://couchdb.apache.org/>>.
14. E. Sit and R. Morris, "Security Considerations for Peer-to-Peer Distributed Hash Tables," Proc.1st Int'l. Wksp. Peer-to-PeerSystems (IPTPS), Cambridge, MA, USA, Mar. 2002.
15. D. Hastorun, M. Jampani, G. Kakulapati, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in ACM Symposium on Operating Systems Principles, 2007.
16. "Twitter's Gizzard." Gizzard: a library for creating distributed datastores. Github Inc., 2010. Web. 20 Oct 2010. <<https://github.com/twitter/gizzard/>>.
17. Mattern, F., Mehl, H., Schoone, A., Tel, G. Global Virtual Time Approximation with Distributed Termination Detection Algorithms. Tech. Rep. RUU-CS-91-32, Department of Computer Science, University of Utrecht, The Netherlands, 1991.
18. Leslie Lamport. 1978. Time, clocks, and the ordering of events in a distributed system. *Communications of ACM* 21, 7 (July 1978), 558-565. DOI=10.1145/359545.359563 <http://doi.acm.org/10.1145/359545.359563>

Vita

Vikram Reddy Kayathi was born in Hyderabad, India in August 1986. He earned his primary and secondary education from Kakatiya Central High School in Hyderabad, Andhra Pradesh. After finishing his high school, he took a very competitive entrance examination for engineering known as EAMCET and stood in top 0.5%. After qualifying this examination he got admission to Department of Information Technology, Chaitanya Bharathi Institute of Technology (CBIT) one of the prestigious institutes in Andhra Pradesh, India. He received his Bachelor of Engineering (B.E.) from Osmania University, Hyderabad, India, in spring 2008. Then he came to United States of America to pursue master's degree. He then joined the graduate program at Louisiana State University, Baton Rouge, in fall 2008. Apart from the academics, he is involved in voluntary activities through Association of India's Development (AID), as a president of Baton rouge, Louisiana, chapter. His career interests are cloud computing, middleware technologies and web technologies, and would like to continue his career in those fields. He is a candidate for the degree of Master of Science in System Science to be awarded at the commencement of fall, 2010.